POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

Mathematics in Machine Learning

Tesina

Credit card default classification

Professors:
Prof. Francesco Vaccarino
Prof. Mauro Gasparini

Authors:
Valerio Zingarelli s281586
Vito Palmisano s288859

Academic Year
2021-2022

Contents

1 Introduction **1**

2 Data Exploration **1**
- 2.1 Dataset Description . . . 1
- 2.2 Data cleaning . . . 3
- 2.3 Data distribution . . . 4
- 2.4 Correlation analysis . . . 7
  - 2.4.1 Numerical features . . . 7
  - 2.4.2 Categorical features . . . 9

3 Data Preprocessing **11**
- 3.1 Pipeline overview . . . 11
- 3.2 Handling Categorical Features . . . 12
- 3.3 Outlier Detection . . . 13
  - 3.3.1 Local Outlier Factor . . . 14
- 3.4 Handling Numerical Features . . . 16
  - 3.4.1 Normalization . . . 16
  - 3.4.2 Standardization . . . 16
- 3.5 Curse of Dimensionality and Dimensionality Reduction . . . 17
  - 3.5.1 Principal Component Analysis . . . 17
  - 3.5.2 Fisher's Linear Discriminant or LDA . . . 19
- 3.6 Resampling . . . 21
  - 3.6.1 Cluster centroid undersampling . . . 21
  - 3.6.2 SMOTE . . . 22
  - 3.6.3 k-means SMOTE . . . 22
  - 3.6.4 SMOTE-NC . . . 23

4 Model Evaluation **24**
- 4.1 Validation . . . 24
- 4.2 K-Fold Cross-Validation . . . 24
- 4.3 Performance Evaluation Metrics . . . 25
  - 4.3.1 Accuracy . . . 25
  - 4.3.2 Precision . . . 25
  - 4.3.3 Recall . . . 26
  - 4.3.4 F1 score . . . 26
  - 4.3.5 ROC curve . . . 26
  - 4.3.6 Precision-Recall curve . . . 27

5 Models Exploration **28**
- 5.1 Linear and Logistic Regression . . . 28
  - 5.1.1 Linear regression in classification tasks . . . 28
  - 5.1.2 Logistic regression . . . 28

5.2 Support Vector Machine 29
5.2.1 Margins and Hard-SVM 29
5.2.2 Fritz John optimality conditions and ”Support Vectors” 30
5.2.3 Soft SVM 30
5.2.4 Kernel’s trick 31
5.3 Decision Trees 33
5.4 Random Forest 34
5.5 KNN 35
5.6 Models Performance comparison 35
5.6.1 Logistic Regression 36
5.6.2 SVM 37
5.6.3 Decision Tree 38
5.6.4 Random Forest 39
5.6.5 K Nearest Neighbors 40

6 Conclusions **41**

1 Introduction

During the work we are exposing in this paper we will explore a dataset named ”Default of credit card clients” using our custom pipeline. Default is something that happens when you have been severely delinquent on your credit card payments. The aim of this work is a binary classification task whose purpose is predict if the client will be in a default situation or not during next month. According to banks' point of view, this fact could affect the possibilities of defaulters users to get future credit card services.

Concerning our work, we will provide a data exploration step, where we will describe the dataset, clean data and analyze the distributions of the features. In addition to that, correlation analysis will be performed for both categorical and numerical features. After that, Data Preprocessing step will follow. In this phase, we will discuss how we decided to deal with possible outliers and with categorical features. In addition to that, dimensionality reduction algorithms will be run in order to simplify computations of our models still mantaining high level in terms of performances. In the end, since we are dealing with an unbalanced dataset, we tried different re-sampling (e.g. undersampling and oversampling) techniques in order to re-balance our classes. In the end, different models have been explored and tested and their results have been compared and analyzed.

At the following link you can find the colab notebook we used to analyze and preprocess the dataset and to train models using the different preprocessing configurations: `https://drive.google.com/file/d/17isR9Oq52wrfMz7wNPQV_SZPSB_UZDh8/view?usp=sharing`

2 Data Exploration

During following sections we will provide a description of the dataset and of its features in order to justify all the choices we will pursue later on.

2.1 Dataset Description

The dataset we are going to deal with during this Tesina work is composed by 30000 different instances of credit card status collected in Taiwan from April 2005 to September 2005. What we want to predict is the class ”default payment next month” that can be 1 in case of default, or 0 on the contrary. Concerning the number of features, we can state there are 23 different features in addition to the target one. We have both numerical and categorical features, so we will have to analyze and deal with them following different approaches. More specifically, the features we are talking about are described by Yeh et al.[1]:

Numerical features

- **LIMIT_BAL**: the limit of the balance that the client can reach.
- **AGE**: age of the clients, expressed in years.

- **BILL_AMT[1,...,6]**: amount of the bill statements from September 2005 (BILL_AMT1) to April 2005 (BILL_AMT6).
- **PAY_AMT[1,...,6]**: amount of the previous payments from September 2005 (PAY_AMT1) to April 2005 (PAY_AMT6).

Categorical Features

- **SEX**: 1 if the credit card owner is a male, 2 if it is a female.
- **EDUCATION**: values from 1 to 4 describing the educational level. More specifically, 1 = graduate school, 2 = university, 3 = high school and 4 = others educational levels.
- **MARRIAGE**: represents the marital status of the credit card owner, 1 = married, 2 = single and 3 = other.
- **PAY_[1,...,6]**: it represents past payments in the period we are considering, from September 2005 (PAY_1) to April 2005 (PAY_6). The measurement scale for the repayment status is: -1 for regular payments, 1 for payment delay of one month, 2 for payment delay of two months, ... , 9 for payment delay of nine months.

ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3
1	20000	2	2	1	24	2	2	-1	-1	-2	-2	3913	3102	689
2	120000	2	2	2	26	-1	2	0	0	0	2	2682	1725	2682
3	90000	2	2	2	34	0	0	0	0	0	0	29239	14027	13559
4	50000	2	2	1	37	0	0	0	0	0	0	46990	48233	49291
5	50000	1	2	1	57	-1	0	-1	0	0	0	8617	5670	35835

BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default payment next month
0	0	0	0	689	0	0	0	0	1
3272	3455	3261	0	1000	1000	1000	0	2000	1
14331	14948	15549	1518	1500	1000	1000	1000	5000	0
28314	28959	29547	2000	2019	1200	1100	1069	1000	0
20940	19146	19131	2000	36681	10000	9000	689	679	0

Figure 1: An extract of our dataset

So, let's see if our dataset reflects what it is stated by Yeh et al.[1]. First of all, in Fig.1, we can see the first five rows of our dataset. At a first glance we can notice that there are some instances taking values equal to -2 for different features PAY_N and this is not reported in the paper, but we will deal with it later. We can see also that the names of the features are the same reported in the paper, exception done for PAY_1, which is not present in our dataset, but we can see that there is a column named PAY_0, so we can think that this is only a typing error and we can consider PAY_0 as PAY_1, so we can rename it. We rename also our target column, from "default payment next month" to "DEFAULT", in order to read it easier during our visualizations.

2.2 Data cleaning

During this phase, the first thing we do is to check if there are null values or empty rows inside our dataset. We can easily state that neither one nor the others are present.

After this check, we go deeper and analyze, for each feature, what are the values it takes and if they are consistent with what Yeh et al. say in [1].

Let's follow the order features have in the dataset, so the first feature we analyze is LIMIT_BAL, which takes 81 unique different values in the whole dataset. Values are something like 20000, 55000 and so on. We do not see nothing strange in this feature. The same is for the SEX feature, which takes only 1 and 2 as values, as we said in the dataset description.

Analyzing the EDUCATION feature, we can see there is something strange, because it takes values ranging from 0 to 6, but based on what we know, it has to take only values between 1 and 4. So we go deeper and notice that there are only few instances having 0, 5 or 6 as value for EDUCATION (respectively 14, 280 and 51 instances). Due to the fact that EDUCATION = 4 stands for 'other educational level', we decide to convert to 4 all the instances having EDUCATION $\in \{0, 5, 6\}$. Let's do the same with the MARRIAGE feature, because also here we have a situation similar to the EDUCATION one. We know that it has to take only 1, 2 or 3 as values, but we can see that there are 54 instances where it takes also 0. As for EDUCATION, we convert to 3 (equivalent to 'other marital status') all the instances having 0 as MARRIAGE value. In Fig.2 we can see how many unique values each categorical feature presents and we can see how, the anomalies we found for the EDUCATION and MARRIAGE features, are related to few instances w.r.t. the whole dataset.

	-2	-1	0	1	2	3	4	5	6	7	8
SEX				11888	18112						
EDUCATION			14	10585	14030	4917	123	280	51		
MARRIAGE			54	13659	15964	323					
PAY_1	2759	5686	14737	3688	2667	322	76	26	11	9	19
PAY_2	3782	6050	15730	28	3927	326	99	25	12	20	1
PAY_3	4085	5938	15764	4	3819	240	76	21	23	27	3
PAY_4	4348	5687	16455	2	3159	180	69	35	5	58	2
PAY_5	4546	5539	16947		2626	178	84	17	4	58	1
PAY_6	4895	5740	16286		2766	184	49	13	19	46	2

(a) Original dataset.

	-1	0	1	2	3	4	5	6	7	8	9
SEX			11888	18112							
EDUCATION			10585	14030	4917	468					
MARRIAGE			13659	15964	377						
PAY_1	8445		14737	3688	2667	322	76	26	11	9	19
PAY_2	9832		15730	28	3927	326	99	25	12	20	1
PAY_3	10023		15764	4	3819	240	76	21	23	27	3
PAY_4	10035		16455	2	3159	180	69	35	5	58	2
PAY_5	10085		16947		2626	178	84	17	4	58	1
PAY_6	10635		16286		2766	184	49	13	19	46	2

(b) Cleaned dataset.

Figure 2: Unique values count for categorical features before and after data cleaning. The features are reported on the rows, while the values they can take are reported on the columns.

Concerning the next feature AGE, it does not show nothing strange, taking values between 21 and 79. But the following features we are going to deal on take abnormal values w.r.t. to what we know from the dataset description. We are talking of all the PAY_i features. We know they have to take values equal to -1 or in the range 1:9, but analyzing the dataset we can see how this is not the case, because they take values in the range -2:8 (Fig.2a). What we think is that there is a shift in the values, so the first thing we do is to re-shift all values, from -2:8 to -1:9, adding 1 to all the values. But we have to do a further consideration: from the description, we know the value 0 is not a possible value, so we have to handle the fact that the re-shift operation transformed -1 vales in 0. We decide to convert all the 0 in -1, following the meaning of this feature, where a value equal to 1 stands for "payment delay of one month", 2 for two months

and so on, so we read 0 as ”payment delay of zero months”, which is equivalent to the meaning of -1 ”regular payments”. So, now we have, as in the description, instances taking values equal to -1 or in the range 1:9 (Fig.2b).

The last features we have to analyze are the ones belonging to the two different types BILL_AMT_i and PAY_AMT_i. In both the cases we do not notice nothing strange, so we end here our data cleaning step and go on analyzing the data distribution.

2.3 Data distribution

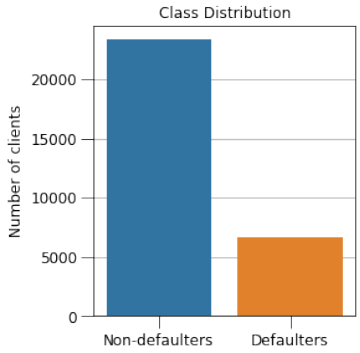

Figure 3: Class balancing. Note our classes are quite unbalanced.

After having described features, let’s have a look at how they are distributed. The first thing we want to look at is the balancing of our dataset. To do that, we have chosen to plot the class distribution. Fig3 displays our classes are quite unbalanced: 80% of the data belongs to non-defaulter class. As a consequence, during the preprocessing steps, we will see how to take some actions, in order to try to mitigate the effects an unbalanced dataset could cause.

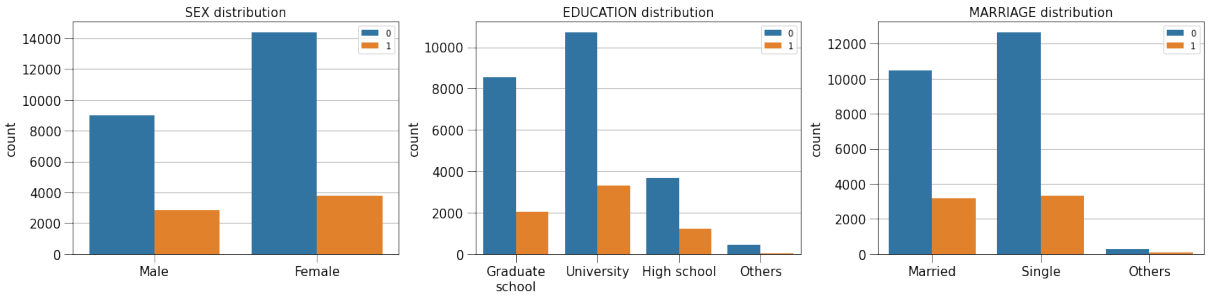

Figure 4: Distribution of categorical features, divided by classes.

In Fig.4 we provide the distributions, divided by classes, of SEX, EDUCATION and MARRIAGE features. We can see how the distributions are well proportioned between the two classes for all the three features, having a ratio quite similar to the 80/20 we saw before.

In Fig.5 we can see the distribution of the features PAY_i. While for values equal to -1 the distributions seems to have a ratio equal to 80/20 between the two classes, for higher values this proportion changes. An instance belonging to defaulters class has a slightly lower probability to have a PAY_i equal to 1, because the ratio seems to be something like 85/15 in this case.

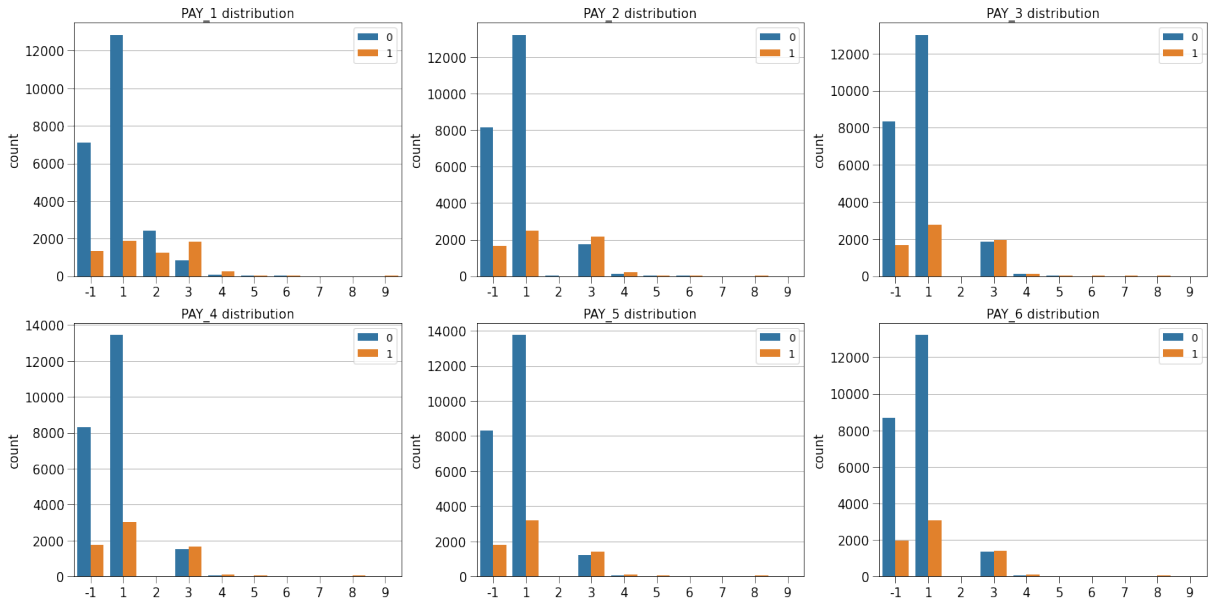

Figure 5: Distribution of PAY_i features.

Instead, for PAY_i values grater then 1, we have an opposite situation, because the ratio seems to decrease, i.e. defaulters have an higher probability to have a PAY_i value grater then 1.
In some cases the ratio takes values 50/50 or 35/65. These observations lead us to consider analyzing the correlation between the target feature and the PAY_i features to be a suitable idea. We manage to find something interesting and next section will provide an overview of such a discovery. Now let's conclude the distribution analysis of the remaining features.

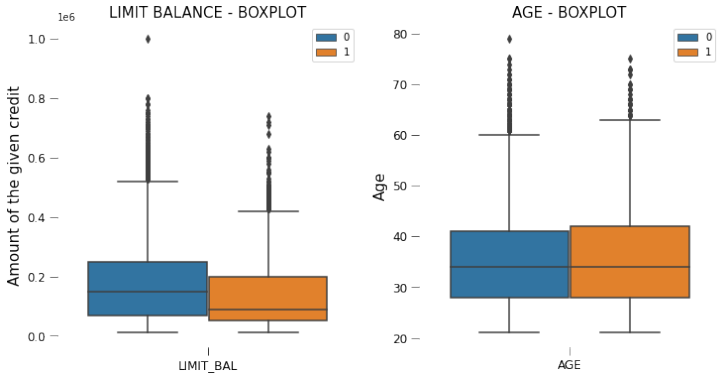

Figure 6: Box plots of LIMIT_BAL and AGE features.

Fig.6 reports the boxplots of the two features LIMIT_BAL and AGE. Concerning the former, it seems that defaulters tend to be users having a slightly lower limit balance w.r.t. not defaulters, but the difference is not so relevant. If we look to the latter we see how the age seems not to be a relevant feature for our task, due to the fact that the AGE data distribution is the same for both defaulters and not defaulters.

In Fig.7 we can compare the boxplots computed separately for the two classes over the BILL_AMT_i features. Independently from the specific BILL_AMT_i we consider, it seems that there is no reason to think that these features will be discriminative for our task. All the boxplots

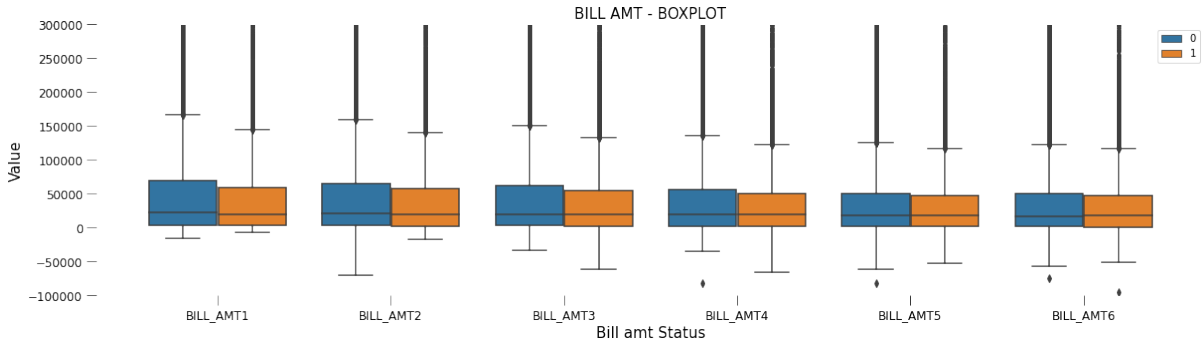

Figure 7: Box plots of BILL_AMT features

have the same median and, for each feature, the quartiles differences between defaulters and not defaulters are neglectable.

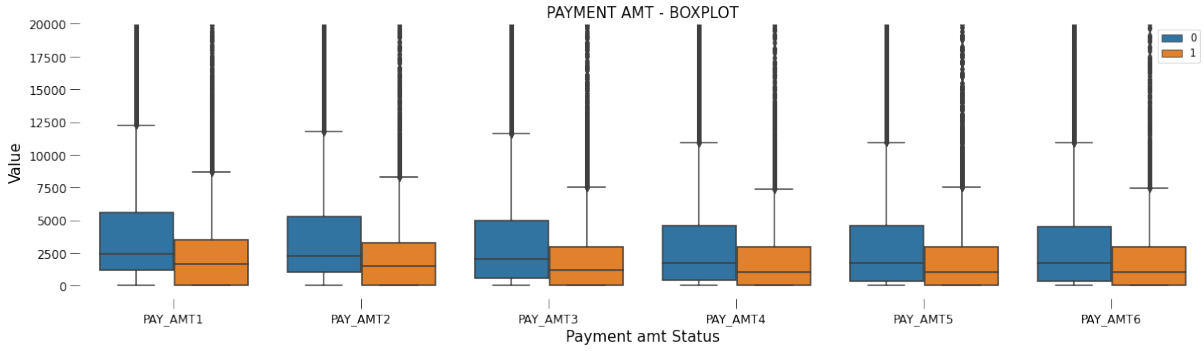

Figure 8: In this figure, box plots of PAY_AMT features are shown. Notice the high quantity of outliers values.

The last distribution we see is the one of Fig.8, where we plot the boxplot of the PAY_AMT_i features. These six comparisons seem to be all quite similar between them. For each feature, the distributions show how the more an user pays in the past months lower is the probability to be a defaulter, i.e. the more a user is up to date with payments, the lower is the probability of being a defaulter.

Now that we concluded the distribution analysis, the next step in our data exploration is the correlation analysis.

2.4 Correlation analysis

In this section we will perform two different correlation analysis, one for numerical predictors and the other for categorical ones.

2.4.1 Numerical features

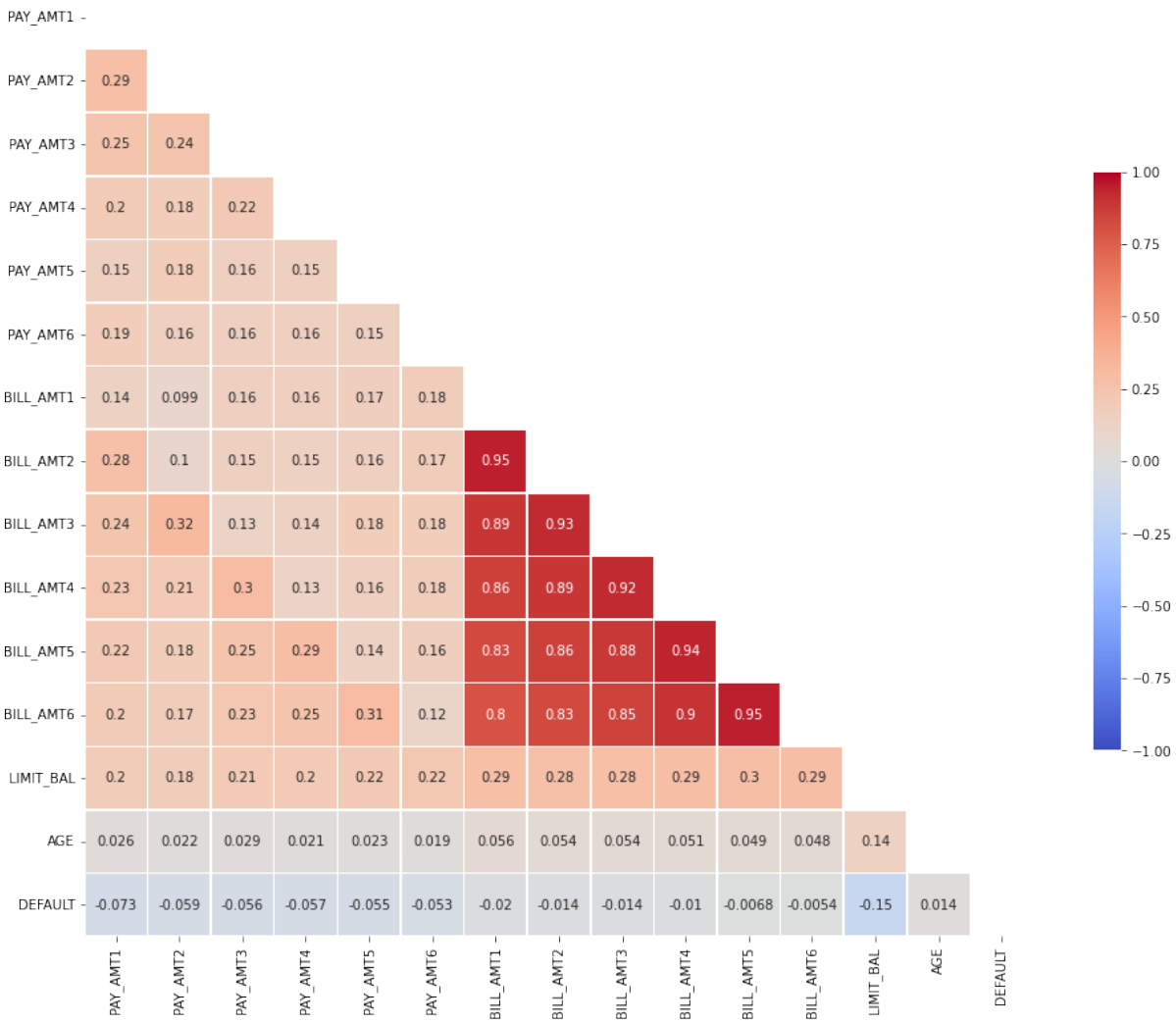

Figure 9: Correlation matrix for numerical values.

Correlation is a statistical measure of how much two variables are linearly related the one with the other. More specifically, if one grows together with the other, we say they have a *positive correlation*; on the contrary, if one grows and the other one becomes smaller, we say they have a *negative correlation*.

The most used correlation metrics is the *Person correlation coefficient*, which tells us how much two variables are linearly related, positively or negatively:

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} \tag{1}$$

Where $Cov(X, Y)$ is the Covariance of random variables X and Y and σ_x and σ_y are their standard deviations. As a matter of fact, this coefficient can only have values $\rho \in [-1, 1]$ and it measures 0 if the random variables X and Y are linearly independent.

Figure 9 displays the linear correlation between the numerical features and also between them and the target feature, computed using the *Person correlation coefficient* as described before. We can notice how no one of the numerical features is linearly correlated with the target one (LIMIT_BAL seems to have a slightly higher negative correlation w.r.t. other features, but it is a negligible value). But all the features are at least slightly positively correlated the one with the other, except for AGE, which is slightly correlated only with LIMIT_BAL.

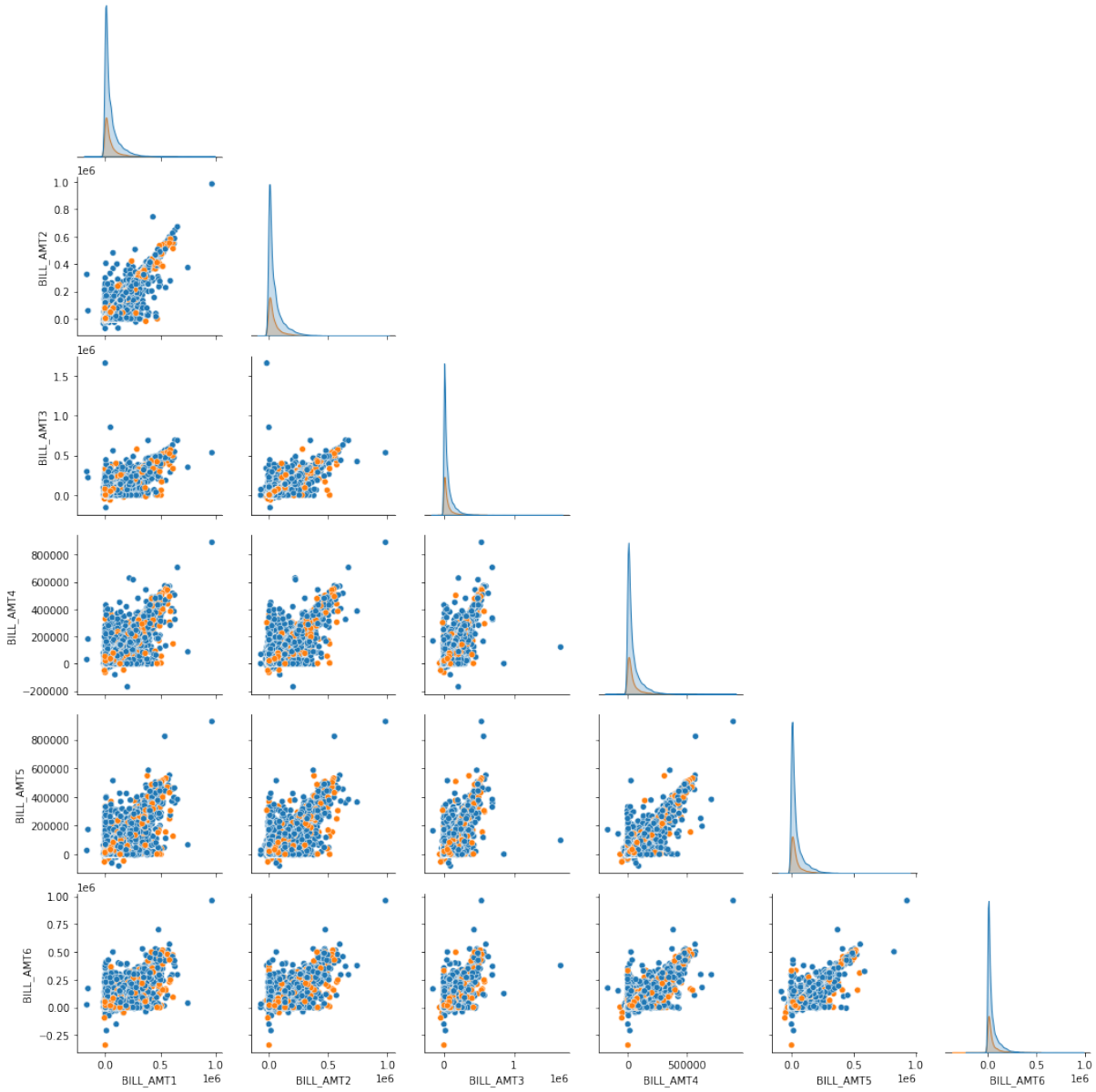

Figure 10: The figure displays a pairplot between the BILL_ATM features

What catches the eye is the strong linear correlation between all the six BILL_AMT_i features. To better analyze this correlation and to see if it exists in both the classes, in Figure.10 we

display, separately for the two classes, all the pairplots between the BIL_AMT features. From the scatterplots we can state that these features are linearly related in the same way for both defaulters and not defaulters, while from the Kernel Density Estimation (KDE) plots on the diagonal (KDE is an estimation of the probability density function without assuming any underlying distribution) we can notice what we have already seen previously, i.e. the distributions of the features are the same for both the classes (in the KDE plots on the diagonal the sum of the areas under the two curves is equal to 1).

2.4.2 Categorical features

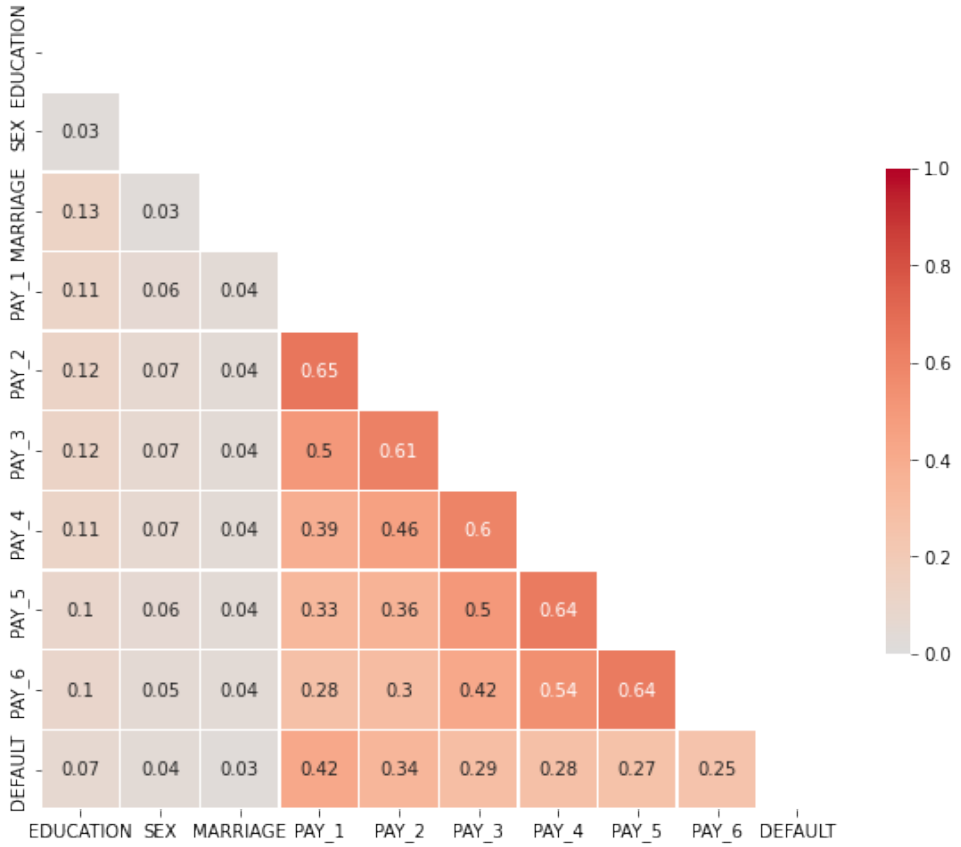

Figure 11: The correlation matrix for categorical features

Concerning categorical features, we decide to use the Cramér's V measure, which sometimes is referred to as Cramér's phi and denoted as φ_c. It is a measure of association between two nominal variables and may be used with variables having two or more levels. Cramér's V varies from 0 (corresponding to no association between the variables) to 1 (complete association) and can reach 1 only when each variable is completely determined by the other. As we can see in the following, it is based on Pearson's chi-squared statistic

$$\varphi_c = \sqrt{\frac{\chi^2}{N(k-1)}} \tag{2}$$

where φ_c is Cramér's V, χ^2 is the Pearson chi-square statistic from chi-square independence test, N is the sample size involved in the test and k is the lower cardinality of either variable.

Figure 11 shows the correlation matrix we got for the categorical features in our dataset. We can notice PAY_j features, for $j \in \{1; 2; ...; 6\}$, are very correlated with each others and with the target DEFAULT too.

From the two correlation matrices we showed, we noticed there are some groups where features are very correlated each other, so, as we will see in the next section, to apply some dimensionality reduction techniques could be a good choice.

3 Data Preprocessing

In this section we describe all the preprocess steps we will perform during our pipeline to transform data. Notice that, for simplicity, we will show transformations fit results applied on the whole train dataset. Indeed, they should slightly different from the real transformations we will compute inside our pipeline. That's because we will perform the fit step inside each k-fold train instead and not on the whole train set, like a fractal.

3.1 Pipeline overview

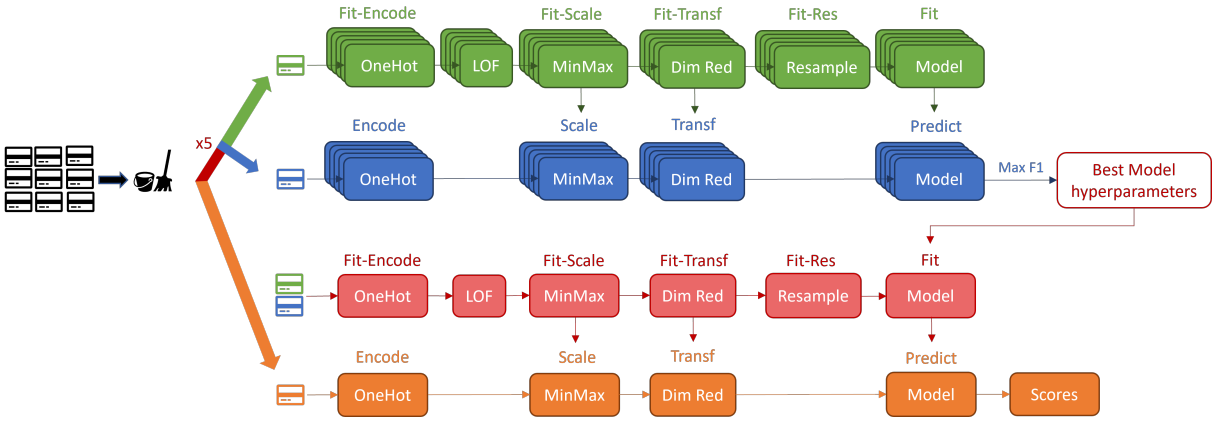

Figure 12: This figure displays a sketch of adopted pipeline. The pink part represents the entire training set, green and blue parts represent K-Fold Cross Validation and the orange boxes are representative of the test set.

Let's now analyze the pipeline we choose to adopt. First of all, data cleaning step is performed. Then, data are splitted according to a 80-20 train-test split and the training set is split again in order to perform Hyper-parameters tuning via K-Fold Cross-Validation. More specifically, our choice is $k = 5$. In each step of Cross-Validation (Green and Blue), the entire pipeline is proposed on the part used for training, like a fractal, as we said before. In the end, Greed-Search is performed in order to get the best hyper-parameters. The model we find this way, is then fitted on the entire training set (after the necessary transformations) and the prediction on test set is performed. In Fig.12 a sketch of the procedure we followed is provided.

3.2 Handling Categorical Features

Machine learning algorithms can only take numerical values, so usually categorical features are not suitable for the model. Usually they need to be transformed somehow. In our case, all the categorical features are already numerical values but, despite that, they are not suitable for the model. That's because using them the way they are would be sub-optimum since these are nominal features, for which it would be wrong to assume an ordering. That's why we decided to adopt *One-Hot Encoding*. In fact, it allows us to remove all the ordering relationships between categorical features adding to the dataset some columns $c_1, c_2, ...c_n$ such that $c_j^i \in \{0; 1\}\ \forall i$, where c_j^i is the i-th element of the added column j. In other words, the idea behind this approach is to create a new dummy feature for each unique value in the nominal feature column. Notice that, if the cardinality of the categorical feature is n, we will only need $n - 1$ new columns to encode its value. Indeed, let's consider the feature "MARRIED" which can have values $v_{\text{MARRIED}} \in \{1; 2; 3\}$. The algorithm will create only the columns "MARRIED_1" and "MARRIED_2", because we can infer the value of "MARRIED_3" thanks to the combination of values of the two generated columns, e.g. the combination "MARRIED_1=0 and MARRIED_2=0" implies that the value of "MARRIED" is 3 because is neither 1 or 2. In the following Fig.13 we can see our dataset after the One-Hot Encoding application.

ID	LIMIT_BAL	AGE	EDUCATION_1	EDUCATION_2	EDUCATION_3	SEX_1	MARRIAGE_1	MARRIAGE_2	PAY_1_-1	...
1	20000	24	0	1	0	0	1	0	0	...
2	120000	26	0	1	0	0	0	1	1	...
3	90000	34	0	1	0	0	0	1	1	...
4	50000	37	0	1	0	0	1	0	1	...
5	50000	57	0	1	0	1	1	0	1	...

Figure 13: Our resulting dataset after the application of our custom get_dummies function and the removal of redundant columns

3.3 Outlier Detection

An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism (Fig.14). Some of the most common causes of outliers are human or technical mistakes. The presence of a significant amount of outliers in some cases could drastically affect the performances. Therefore, it is a common practise to train the model with and without them in order to catch their contribution.

Figure 14: A very intuitive idea of outlier concept: the white bottle has a very different color and shape with respect to other bottles.

We can distinguish among three different kind of outliers (Fig.15):

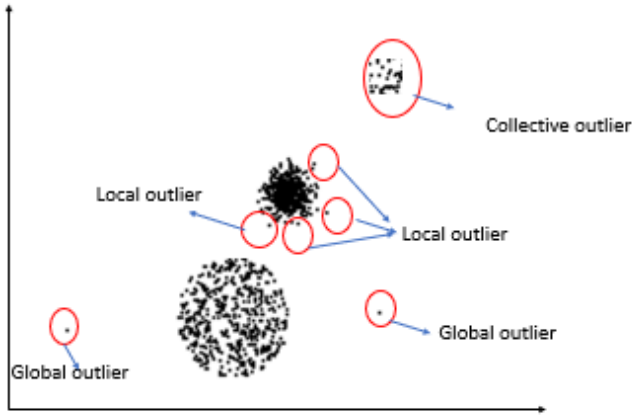

Figure 15: The three different kind of outliers.

- **Global outliers:** data instances that deviate significantly from the rest of the dataset, referred to as point anomalies. Global outliers are the simplest type of outliers addressed by most outlier detection algorithms.
- **Contextual or Conditional outliers:** data instances that appear to have a value within the normal range for the entire dataset. Still, they are unusually different when a context is applied or when you look at the surrounding data points. Sometimes they are also called Local Outliers.

- **Collective Outliers:** data instances that deviate significantly from the entire dataset, but individual data instances may not be an outlier.

There are several different techniques for removing outliers: graphical techinques, such as *boxplots*, or algorithm such as Local Outlier Factor [2].

3.3.1 Local Outlier Factor

The local outlier factor is based on a concept of a local density, where locality is given by k nearest neighbors, whose distance is used to estimate the density. By comparing the local density of an object to the local densities of its neighbors, one can identify regions of similar density, and points that have a substantially lower density than their neighbors. These are considered to be **outliers**. Let $k\text{-}distance(A)$ be the distance of the object A to the k-th nearest neighbor. This distance is used to define the so called reachability distance $d_r^k(A;B)$

$$d_r^k(A;B) = max\{k\text{-}distance(B); d(A;B)\} \tag{3}$$

In other words, $d_r^k(A;B)$ is the true distance of the two objects, but at least the k-distance of B, as we can see in Fig.16. Starting from this formula, we can state that all the k nearest points of B will be treated as the same.

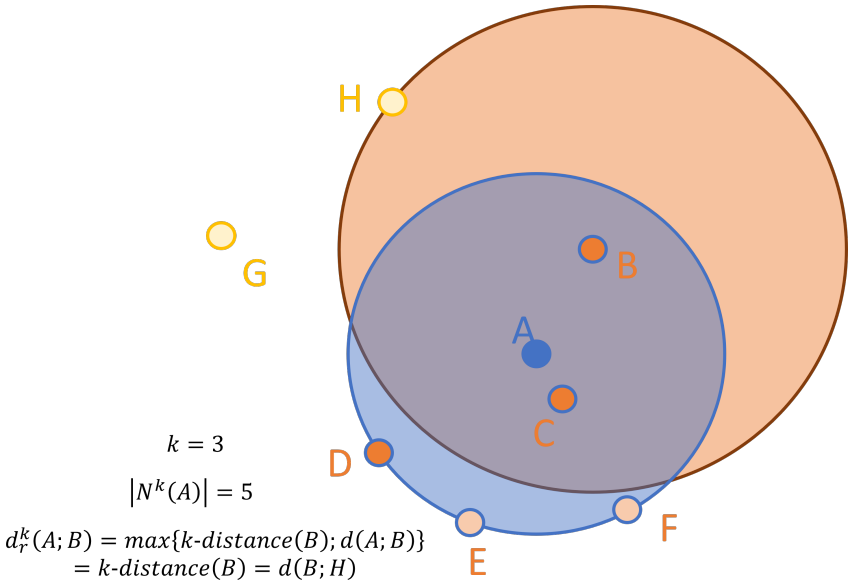

Figure 16: Reachability distance computation example idea.

Let's now define the *local reachability distance* $ld_r^k(A)$ which is the inverse of the average reachability distance of the object A from its neighbors.

$$ld_r^k(A) := \frac{1}{\frac{\sum_{A' \in N_k(A)} d_r^k(A;A')}{|N_k(A)|}} = \frac{|N_k(A)|}{\sum_{A' \in N_k(A)} d_r^k(A;A')} \tag{4}$$

In the end, the local reachability distances are compared with those of the neighbors, according to:

$$LOF(A) = \frac{\sum_{A' \in N_k(A)} ld_r^k(A')}{|N_k(A)| * ld_r^k(A)} \tag{5}$$

Basing on the value of LOF the points are classified as outliers or not according to:

- $LOF \approx 1$: Similar density as neighbors
- $LOF < 1$: Higher density then neighbors
- $LOF > 1$: Outlier

After applying such an algorithm on the whole dataset, we found out it managed to catch 6552 outliers. Of course this is a quite high value and, therefore, we reasoned on it. In the end, we realized that inside the dataset we are dealing with, it is not strange to have situations like users having very high balance limits, or having very high bill statement amounts, because there could be people much richer than the average. We thought also that defaulters are not quite common users, so, maybe, performing outlier removal we are removing what we want to look for instead! We so tried to check if the outliers we identified belongs the more to the DEFAULT class or not and we found that the unbalanced proportion 80/20 we have in the dataset is present also in the subset of outliers we identified. Due to all these observations, we decided to perform both the computations inside our pipeline, with and without outliers removal.

3.4 Handling Numerical Features

Feature scaling is one of the most important data preprocessing step in machine learning. Algorithms that compute the distance between the features are biased towards numerically larger values if the data is not scaled. Also, feature scaling helps machine learning, and deep learning algorithms train and converge faster.

3.4.1 Normalization

Normalization or Min-Max Scaling is used to transform features to be on the same scale. The new point is computed as:

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{6}$$

In this way we force data between 0 and 1. Normalization is useful when there are no outliers as it cannot cope up with them.

3.4.2 Standardization

On the other side, standardization or Z-Score Normalization is the transformation of features by subtracting from mean and dividing by standard deviation:

$$X_{new} = \frac{X - \mu}{\sigma} \tag{7}$$

Note that, since there is not a predefined range of transformed features, standardization is not affected by outliers as much as the Min-Max Scaler.

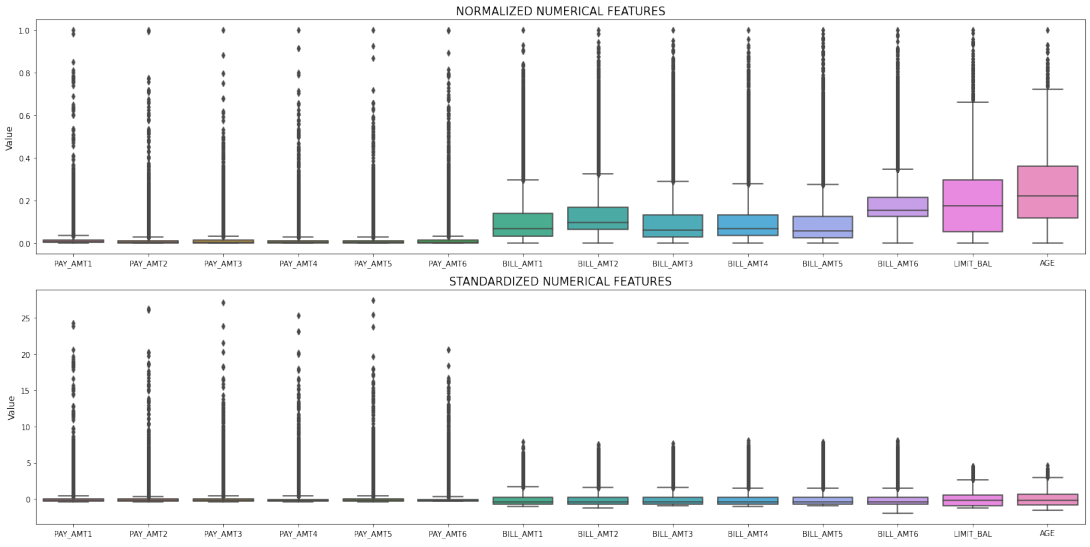

Figure 17: Comparison between standardized and normalized features.

Since we perform outlier removal, we decided to adopt normalization. In fact, normalization is affected by outliers'presence but the appliance of LOF allowed us to avoid such a problem. In addition, even when we do not perform outliers'removal, we apply normalization as well. That's for the same reasons we exposed in the previous section.

3.5 Curse of Dimensionality and Dimensionality Reduction

The Curse of Dimensionality is a well-known problem in machine-learning and big-data enviroment and it refers to when our data has too many features. Without going deep in maths, a very good visual example is provided in Fig.18.

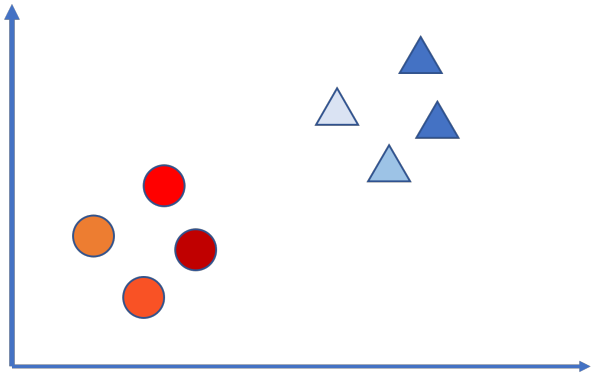

Figure 18: A graphical idea of curse of dimensionality.

From the figure, it's easy to notice how the points are well separated according to their color. In fact, red points are together and so do blue points. But, if we don't consider the feature "color" to be binary (i.e. blue or red), and consider, instead, all of the possible shades of colors (light blue, magenta,..), a clustering algorithm will fail to cluster points according to their color [3]. In addition to that, we can immagine that the more dimensionality increases, the more the volume of the spaces increases as well and that's could be a problem because data become more and more sparse. Because of this, computational complexity problems may occur. In order to solve such an issue, several algorithms have been tested and invented during the years. In our analisys, we will present and use 2 of them: PCA and Fisher's Linear Discriminant, also called LDA.

3.5.1 Principal Component Analysis

Let's now have a look at Principal Component Analysis (from now on PCA). Consider $\boldsymbol{x_1}, ..., \boldsymbol{x_m} \in R^d$, we would like to reduce the dimensionality of these vectors (from d to n) using a linear transformation and, in order to do that, we introduce a mapping $\boldsymbol{x} \rightarrow W\boldsymbol{x}$, where $W\boldsymbol{x}$ is the reducted dimensionality representation of $\boldsymbol{x}$. Of course, we may like to do the opposite and so we introduce $U \in R^{d,n}$ which is a matrix that can recover $\boldsymbol{x}$ from its reduction. The problem we want to solve, for PCA, is:

$$\underset{W \in R^{d,n} U \in R^{d,n}}{\operatorname{argmin}} \sum_{i=1}^{m} \|\boldsymbol{x_i} - UW\boldsymbol{x_i}\|_2^2 \tag{8}$$

Considering the fact that if (U, W) is a solution to (8), the columns of U are orthonormal and $W = U^T$, the problem becomes:

$$\underset{U \in R^{d,n} UU^T = I}{\operatorname{argmin}} \sum_{i=1}^{m} \left\|\boldsymbol{x_i} - UU^T\boldsymbol{x_i}\right\|_2^2 \tag{9}$$

Let's now try to go further, generally:

$$\left\|\boldsymbol{x} - UU^T\boldsymbol{x}\right\|^2 = \|\boldsymbol{x}\|^2 - trace(U^T\boldsymbol{x}\boldsymbol{x}^{\boldsymbol{T}}U) \tag{10}$$

where $trace$ is the sum of the diagonal values of a matrix. Then, we can substitute and we obtain the final form of the problem:

$$\underset{U \in R^{d,n}: UU^T = I}{\operatorname{argmax}} \quad trace\left(U^T \sum_{i=1}^{m} \boldsymbol{x_i}\boldsymbol{x_i^T}U\right) \tag{11}$$

Let's call $A = \sum_{i=1}^{m} \boldsymbol{x_i}\boldsymbol{x_i^T}$, the *Scatter Matrix*. A is symmetric and so can be decomposed $A = VDV^T$ where D is a diagonal matrix which values are the eigenvalues of A and $VV^T = I$. Assuming D is positive semidefinite, indeed $D_{1,1} \geq ... \geq D_{d,d}$, we can state that the solution to (11) is the matrix U whose columns are the n eigenvectors of A corresponding to the largest n eigenvalues. But, besides computational problems, why should we use such a tool? Well, PCA creates a set of principal components that are rank ordered by variance (the first component has higher variance than the second, the second has higher variance than the third, and so on), uncorrelated, and low in number (we can throw away the lower ranked components). In this way, we will have an "ideal" set of features because they will:

- **have high variance**: in such a way, the model could generalize much more efficiently.
- **be uncorrelated**: since the principal components are orthonormal, they will be uncorrelated as well
- **be not many**: having fewer features helps the model to avoid overfitting

Concerning the number of components, we decided to adopt $n_{components} = 26$ because they managed to catch more then 99% of the variance as Fig.19 displays.

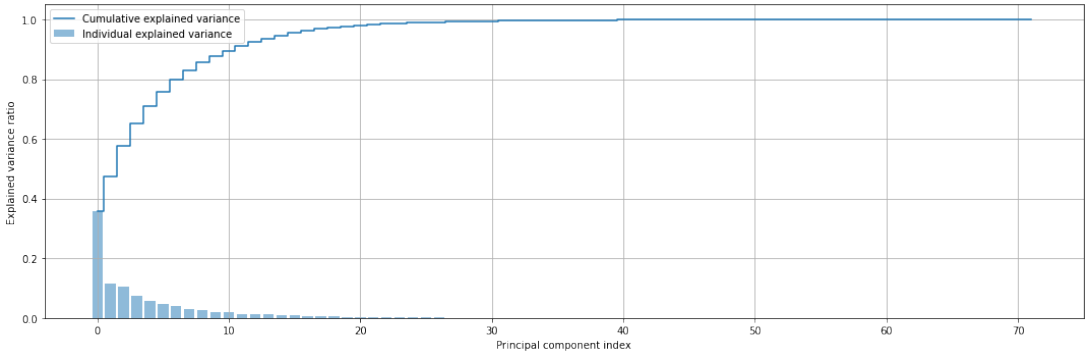

Figure 19: Variance explained

3.5.2 Fisher's Linear Discriminant or LDA

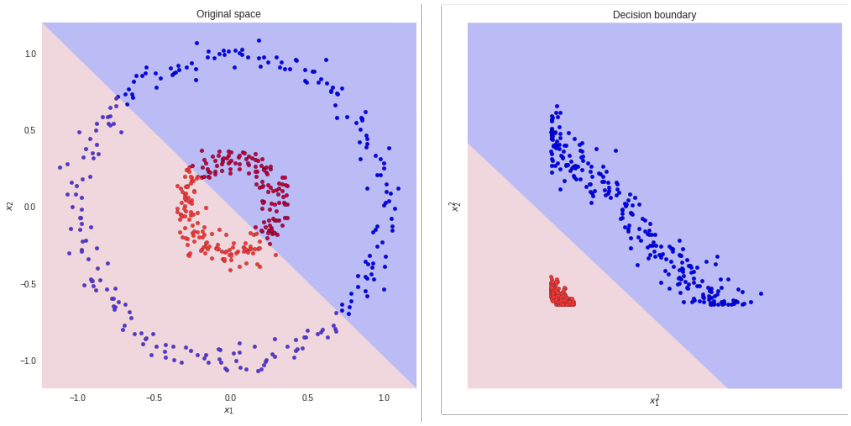

Figure 20: Main idea behind LDA. Classes are separed using variance between classes and variance within class.

Another possible implementation of a dimensionality reduction algorithm is Linear Discriminant Analysis, from now on LDA. The idea behind LDA is simple. Mathematically speaking, we need to find a new feature space to project the data in order to maximize classes separability (Fig.20). Differently from PCA, LDA takes into account the categories in the data. The transformation is based on maximizing the ratio of "between-class variance" to "within-class variance" with the goal of reducing data variation in the same class and increasing the separation between classes. Let's now try to be more formal. Consider $S = (\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_m}, y_m)$ to be our training set, composed by N_0 points belonging to class C_0 and N_1 points belonging to class C_1. We define the mean of the points with label k as:

$$\boldsymbol{\mu_k} = \frac{1}{N_k} \sum_{\boldsymbol{x} \in C_k} \boldsymbol{x} \ , \quad k = 0, 1 \tag{12}$$

Now, let's consider the projection of the sample $\boldsymbol{x}$ onto a line in direction $\boldsymbol{v}$, it is given by $\boldsymbol{v^T x}$. The question is, how to measure separation between projections of different classes? Given $\boldsymbol{\mu_k}$, we can compute $\tilde{\mu}_k$, the mean of the projection of class k over the direction given by $\boldsymbol{v}$, as:

$$\tilde{\mu}_k = \frac{1}{N_k} \sum_{\boldsymbol{x} \in C_k} (\boldsymbol{v^T x}) = \boldsymbol{v^T \mu_k} \tag{13}$$

The quantity $|\tilde{\mu}_0 - \tilde{\mu}_1|$ seems to be a good measure for the separation between the projections of the two classes. But it does not consider the variance of the classes, so let's normalize it by a factor which is proportional with the variance: the scatter of the projected samples! The scatter measures the *variance within class* for class k. It measures the spread of data around the mean, the same that variance do, but scatter is just on different scale than variance:

$$\tilde{s}_k^2 = \sum_{\boldsymbol{x} \in C_k} (\boldsymbol{v^T x} - \tilde{\mu}_k)^2 \tag{14}$$

We can now maximize the ratio of "between-class variance" to "within-class variance"

$$J(\boldsymbol{v}) = \frac{|\tilde{\mu}_0 - \tilde{\mu}_1|}{\tilde{s}_0^2 + \tilde{s}_1^2} \tag{15}$$

All we need to do now is to express J explicitly as a function of $\boldsymbol{v}$ and maximize it. But before, we have to define the separate class scatter matrices S_1 and S_2 for classes 1 and 2.

$$S_k = \sum_{x \in C_k} (\boldsymbol{x} - \boldsymbol{\mu_k})(\boldsymbol{x} - \boldsymbol{\mu_k})^T \,, \quad k = 0, 1 \tag{16}$$

These measure the scatter of original samples x_i (before projection). After we define the within the class scatter matrix S_W

$$S_W = S_1 + S_2 \tag{17}$$

Now, we define the between the class scatter matrix S_B, which measures the separation between the means of two classes (before projection).

$$S_B = (\boldsymbol{\mu_1} - \boldsymbol{\mu_2})(\boldsymbol{\mu_1} - \boldsymbol{\mu_2})^2 \tag{18}$$

Thus our objective function can be rewritten after some computations:

$$J(\boldsymbol{v}) = \frac{|\tilde{\mu}_0 - \tilde{\mu}_1|}{\tilde{s}_0^2 + \tilde{s}_1^2} = \frac{\boldsymbol{v}^T S_B \boldsymbol{v}}{\boldsymbol{v}^T S_W \boldsymbol{v}} \tag{19}$$

The last step is to find $\boldsymbol{v}^*$

$$\boldsymbol{v}^* = \underset{\boldsymbol{v}}{\operatorname{argmax}}\, J(\boldsymbol{v}) = \underset{\boldsymbol{v}}{\operatorname{argmax}}\, \frac{\boldsymbol{v}^T S_B \boldsymbol{v}}{\boldsymbol{v}^T S_W \boldsymbol{v}} \tag{20}$$

So, we aim to find the direction which minimises the *within-class variance* and maximises the *between-class variance*.

3.6 Resampling

In Fig 3., we showed how our dataset is really imbalanced. When a model learns from an unbalanced dataset, it is very hard to reach a very high level performances. In fact, we could achieve almost 80% accuracy by just predicting the majority class (non-defaulters) for all examples, in our case. In order to fix this lack of generality, we can apply some *resempling techniques* to balance the classes. The most naive idea is simply to delete data from the majority class (or duplicate data from minority class) since the desired level is obtained. Of course, this kind of approach is discouraged because, random undersampling does not allow to control which information is discarded, on the other hand random oversampling causes overfitting because the model is trained on many identical data. To solve or, at least, mitigate these issues, we choose to test the Cluster Centroid method, the Synthetic Minority Oversampling Technique (SMOTE) and some variations of the latter. Fig 21 displays both general re-sampling techniques.

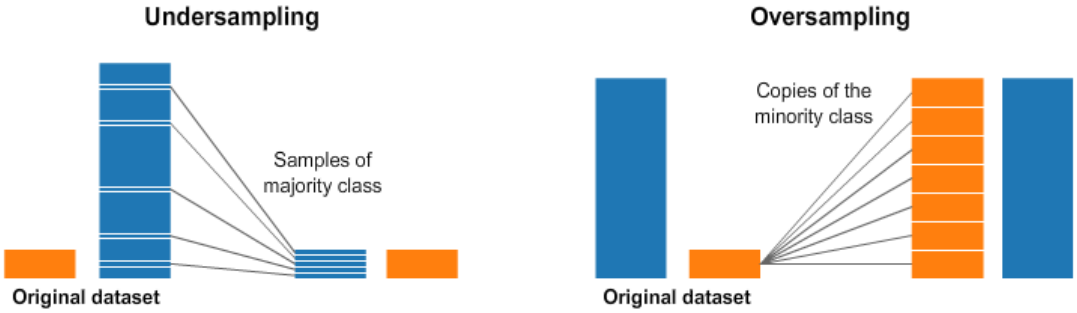

Figure 21: An idea of re-sampling. Sub-figure on the right displays Oversampling and, on the other side, sub-figure on the left represents undersampling

3.6.1 Cluster centroid undersampling

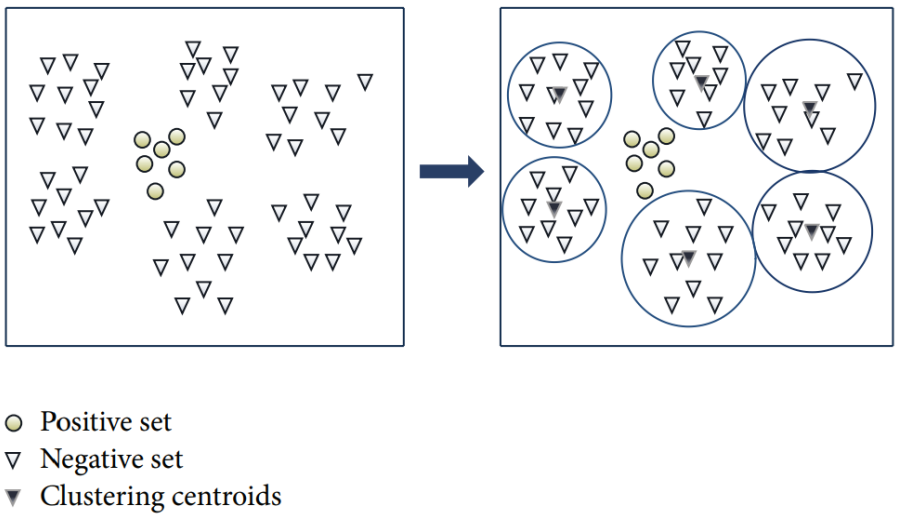

Figure 22: Cluster Centroid Undersampling

The worst problem regarding undersampling, is the loss of generality deriving from discarded data belonging to majority class. Something that can help us to overcome this issue, is Cluster Centroid method for undersampling. More specifically, it replaces clusters of majority samples with the respective cluster centroids. A K-means algorithm is fitted to the data, then, the majority of samples from the clusters are entirely substituted by the sets of cluster centroids from K-Means [4]. Cluster Centroids contain the most representative variations of the majority class in which features values would be visualized at the center. Fig 22 displays an idea of the algorithm.

3.6.2 SMOTE

Figure 23: Steps of SMOTE algorithm

Synthetic Minority Oversampling Technique is the most used over-sampling technique. The main idea behind SMOTE is to create or synthesize elements or samples from the minority class rather than creating copies based on those that exist already. This is used to avoid model overfitting. Indeed, we take two points of the minority class and compute the features of the synthetic point as the mean of the features of the original points. In Fig.23 we can see the steps we described. Notice that such an algorithm is not suitable with OneHotEncoding preprocessing. Concerning implementation, we decided to use the function "SMOTE" from *imbalanced learning*.

3.6.3 k-means SMOTE

Felix Last, Georgios Douzas, and Fernando Bacao proposed in their paper an oversampling approach which is a combination of SMOTE technique and k-means clustering algorithm[5]. They wanted to solve the issue with SMOTE algorithm, in particular the noise that can be amplified applying such an oversampling technique and they succeed to avoid such noise adding k-means to SMOTE. Moreover, k-means SMOTE's focus is placed on both between-class imbalance and within-class imbalance, fighting the small disjuncts problem by inflating sparse minority areas. The main steps of the algorithm are:

1. Input space is clustered into k groups using k-means clustering.

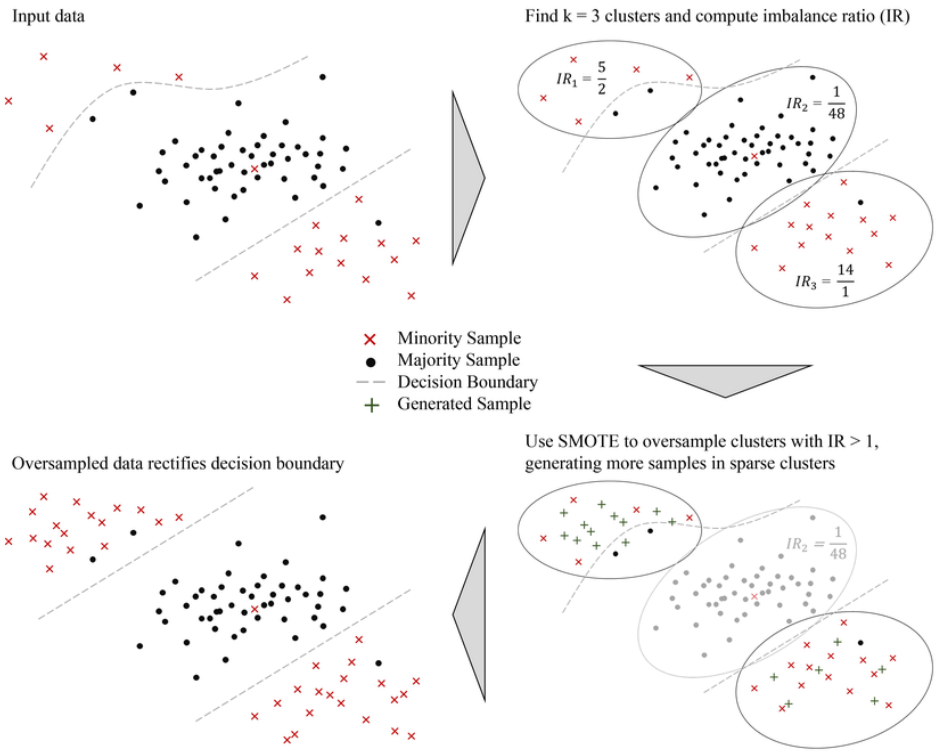

Figure 24: The main steps of k-means SMOTE algorithm

2. The filtering step selects clusters for oversampling, retaining those with a high proportion of minority class samples. It then distributes the number of synthetic samples to generate, assigning more samples to clusters where minority samples are sparsely distributed.

3. In the end, SMOTE is applied in each selected cluster to achieve the target ratio of minority and majority instances. In Fig.24 the main steps of the algorithm are displayed.

3.6.4 SMOTE-NC

As we mentioned before, SMOTE interpolates the points of minority class in order to syntethize new points. Acting like that, it could generate some errors in categorical features because it would use the columns of One-Hot Encoding to generate points. In such a way, categorical features will be not categorical anymore. In order to find a solution to this issue, we decided to use an efficient variant of SMOTE whose aim is to make the application of SMOTE technique feasible even for categorical features: *SMOTE-NC*. SMOTE-NC slightly change the way a new sample is generated by performing something specific for the categorical features. In fact, the categories of a new generated sample are decided by picking the most frequent category of the nearest neighbors present during the generation.

4 Model Evaluation

4.1 Validation

In order to obtain an accurate estimation of the true risk, we use some of the training data as a validation set, over which one can evaluate the success of the algorithm's output predictor. This procedure is called *validation*. This kind of approach can be used to for model selection: we first train different algorithms (or the same algorithm with different parameters) on the given training set. Let $H = \{h1; ...; h_r\}$ be the set of all output predictors of the different algorithms. Now, to choose a single predictor from H we sample a fresh validation set and choose the predictor that minimizes the error over the validation set. It can be demonstrated (Theorem 11.2 from [6]) that the error on the validation set approximates the true error as long as H is not too large. However, if we try too many methods (resulting in $|H|$ that is large relative to the size of the validation set) then we're in danger of overfitting.

4.2 K-Fold Cross-Validation

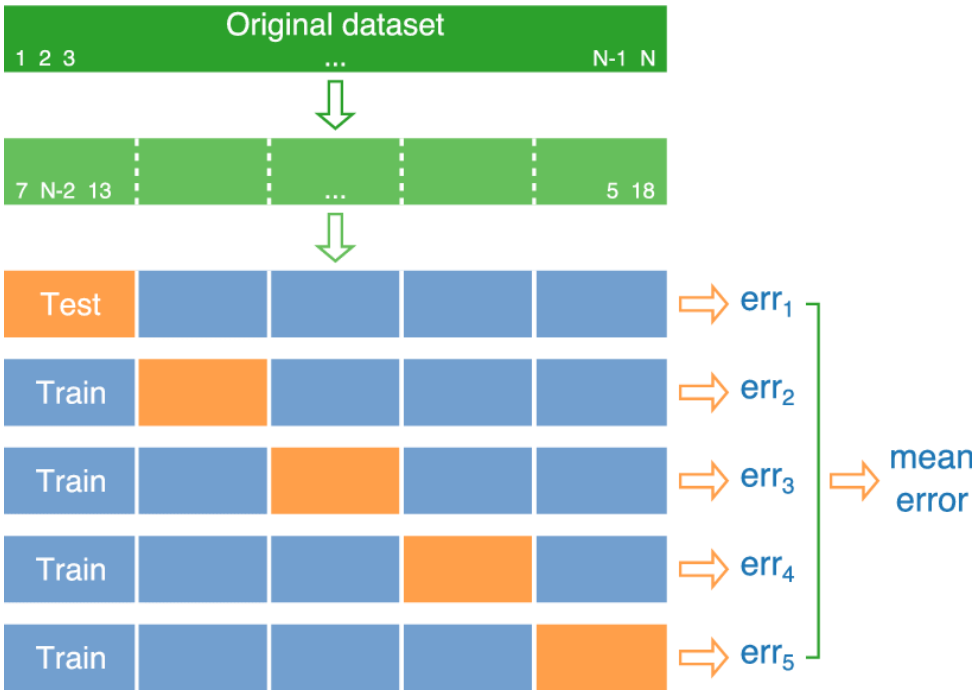

Figure 25: Main idea behind cross validation

Sometimes it happens we have very few data and we can't use many to validation. In such cases, *k-fold cross validation* may help us. In k-fold cross validation the original training set is partitioned into k subsets (folds) of size $\frac{m}{k}$ (for simplicity, assume that $\frac{m}{k}$ is an integer). For each fold, the algorithm is trained on the union of the other folds and then the error of its output is estimated using the fold. Finally, the average of all these errors is the true error (Fig.25). Once we have found satisfactory hyperparameter values, we can retrain the model on the complete training dataset and obtain a final performance estimate using the independent

test dataset. Since k-fold cross-validation is a resampling technique without replacement, the advantage of this approach is that each example will be used for training and validation (as part of a test fold) exactly once, which yields a low-variance estimate of the model performance. In fact, if the model sees the same data more then once, they will become part of the training set and performances may become poor.

Well, now let's wonder how to choose the correct value of k. Beside k is an hyper-parameter like the others, some studies have displayed which are the best values to adopt. According to [7], one of the best values is $k = 10$ but, in case of big datasets (like the one we have adopted), even smaller values are correct. Moreover, it's important to specify that a good practice is to apply stratification in order to check if all the folds are representative for both classes.

4.3 Performance Evaluation Metrics

In this section we will provide a description of the metrics we decided to use for evaluating our model's performances.

4.3.1 Accuracy

Accuracy is the most simple metrics one can use in classifications tasks. Basically it is the number of correct classified items over the total number of items.

$$Accuracy = \frac{\#correct\ classified\ items}{\#total\ items} = \frac{TP + TN}{TP + TN + FN + FP} \tag{21}$$

Where the abbreviations stand for:

- TP: True Positive. The number of items predicted as belonging to Positive class and, effectively, they belong to it
- FP: False Positive. The number of items predicted as belonging to Positive class but they belong to Negative one
- TN: True Negative. The number of items predicted as belonging to Negative class and, effectively, they belong to it
- FN: False Negative. The number of items predicted as belonging to Negative class but they belong to Positive one

Using the concepts expressed before, we can derive some other metrics:

4.3.2 Precision

Precision is the fraction of correctly predicted Positive items over the total number of predicted positive items.

$$Precision = \frac{TP}{TP + FP} \tag{22}$$

4.3.3 Recall

Recall is the fraction of correctly predicted Positive items over the total number of positive items.

$$Recall = \frac{TP}{TP + FN} \tag{23}$$

4.3.4 F1 score

F1 score is a combination between the former expressed metrics. More specifically is defined as:

$$F1 = 2 * \frac{PrecisionRecall}{Precision + Recall} = \tag{24}$$

4.3.5 ROC curve

A ROC (receiver operating characteristic) curve is a plot of how well a binary classifier performs. It considers $Recall$ on one axis against the false positive rate (FPR) on the other, where FPR is defined as:

$$FPR = \frac{FP}{FP + TN} \tag{25}$$

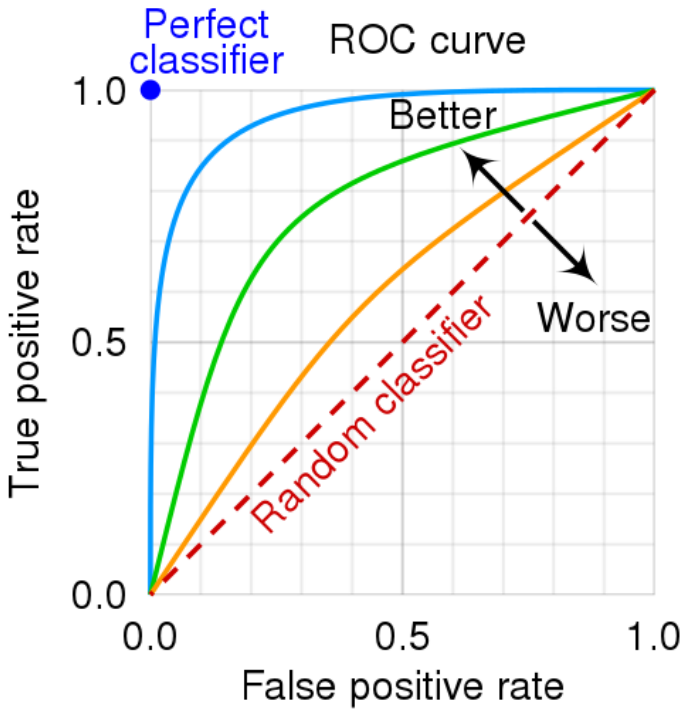

Figure 26: An example of ROC curve

In Fig.26, an idea of ROC curve is provided. Moreover, according to [8], although ROC graphs are widely used to evaluate classifiers under presence of class imbalance, it has a drawback: under class rarity, that is, when the problem of class imbalance is associated to the presence of a low sample size of minority instances, as the estimates can be unreliable. A suitable alternative graph is Precision-Recall curve.

4.3.6 Precision-Recall curve

In a binary classification problem, we may predict if the class is positive or negative, or, alternatively, the probabilities for each class. The reason behind this is to provide the capability to choose and even calibrate the $threshold$ for predicted probabilities. For example, a default might be to use a threshold of 0.5, meaning that a probability in $[0.0, 0.49]$ is a negative outcome, and a probability in $[0.5, 1.0]$ is a positive outcome. This threshold can be calibrated in order to adapt the behavior of the model for the specific problem we are dealing with right now. Basing on this concept, we can define a Precision-Recall curve according to the following.
A precision-recall (PR) curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds. We use it to explore the trade-off between the well-classified positive examples and the number of mis-classified negative examples. Besides, the area under the PR curve (AUC) is a good way to get a score for the general performance of a classifier and to compare it to the one of another classifier. Noting that the PR space is a unit square, it can be clearly seen that the AUC for a classifier f is such that $AUC(f) \in [0, 1]$, with the upper bound attained for a perfect classifier (one with p=1 and r=1).

5 Models Exploration

5.1 Linear and Logistic Regression

5.1.1 Linear regression in classification tasks

When we deal with classification tasks, we may wonder "What is the relationship between $P(Y = 1|X)$ and X itself?". Well, one possible answer to this question is linear regression.

$$p(x) = \beta_0 + \beta_1 x \tag{26}$$

The most important drawback of such an approach is that using it we could get results out of the range [0,1] and this would make hard to read such results as probabilities. That's why we introduce Logistic Regression.

5.1.2 Logistic regression

Logistic Regression classifies the response variable by giving back the probability that the given point belongs to a specific class. In fact, in logistic regression, we learn a family of functions h from R^d to interval [0,1], that can be interpreted as the probability of a point $\boldsymbol{x}$ to belong to class 1. The sigmoid (i.e. "S-shaped") function which is used in Logistic regression is the *logistic function*, defined as follows:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{27}$$

This function is able to model the probability a point has to belong to a certain class. In fact, using a threshold (typically 0.5) one can classify a data point as belonging to class 0 or 1. Combining (26) and (27) we can write:

$$p(x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}} \tag{28}$$

Considering the multiple explanatory variables case, if we divide by $(1 - p(\boldsymbol{x}))$ and apply logarithm to both members, we get:

$$log\left(\frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})}\right) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \tag{29}$$

Note that the left part of equation is called $logit$ or *log odds* [9]. Now the problems are: how we evaluate such a function? How to choose/compute the best βs? Well, we need to find an appropriate loss function for this model. In other words, we should define how bad it is to predict some $h_{\boldsymbol{w}}(\boldsymbol{x}) \in [0, 1]$ given that the true label is y $\in \{0, 1\}$. To do that, the best loss choice is:

$$L(\sigma(\boldsymbol{x}); y) = \begin{cases} -log(p(\boldsymbol{x})) & \text{if y=1} \\ -log(1 - p(\boldsymbol{x})) & \text{if y=0} \end{cases} \tag{30}$$

A generalization of this concept is the *Binary Cross Entropy Loss*, which can be written as:

$$l_{BCE} = -\frac{1}{m}\left[\sum_{i=1}^{m} y_i log(p(\boldsymbol{x}_i)) + (1 - y_i) log(1 - p(\boldsymbol{x}_i))\right] \tag{31}$$

5.2 Support Vector Machine

Support Vector Machines are a very useful machine learning algorithm. The main concept behind them is to find the *best separating hyper-plane*. Why the "best"? Well, as it's possible to imagine, there are infinite hyper-plane capable of separating our data, but, roughly speaking, the best one is the one that maximizes margins. An intuitive idea is in the following Fig.27.

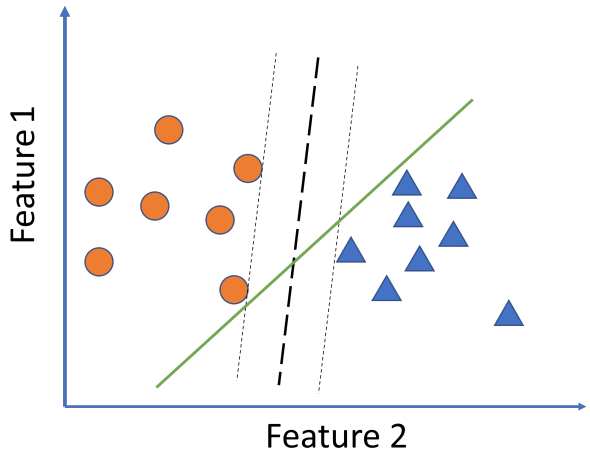

Figure 27: An explanatory graphical example of SVM

Both the hyper-planes represented by the green and black lines manage to separate well our data but the former has not a "large" margin. The latter, instead, maximizes such margins. Let's now try to be more formal and add some definitions.

5.2.1 Margins and Hard-SVM

Let $S = (\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_m}, y_m)$ be our training set and be $\boldsymbol{x_i} \in R^d$ and $y_i \in \{-1, 1\}$; we say that S is *linearly separable* if there exists a half-space $(\boldsymbol{w}, b)$ such that $y_i = sign(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b)$ for all i, or, alternatively:

$$\forall i \in [m],\ y_i(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b) > 0 \tag{32}$$

We talk about *Hard-SVM* when we do not let the points lie within the boundaries. A more formal mathematical formulation of the problem of *Hard-SVM* is:

$$\underset{(\boldsymbol{w},b):\|\boldsymbol{w}\|=1}{\operatorname{argmax}} \min_{i\in[m]} |\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b| \quad s.t. \quad \forall i,\ y_i(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b) > 0 \tag{33}$$

In other words, *Hard-SVM* is the learning rule that gives us back the hyper-plane that separates the training set with the largest possible margin. Anyway, the *Hard-SVM* rule, could be formulated also in a quadratic-optimization form:

$$(w_0, b_0) = \underset{(\boldsymbol{w},b)}{\operatorname{argmin}} \|\boldsymbol{w}\|^2 \quad s.t. \quad \forall i,\ y_i(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b) \geq 1 \tag{34}$$

Finding the solution to the problem exposed in (34), we can compute an optimal solution for (33)

$$(\hat{\boldsymbol{w}}, \hat{b}) = \left(\frac{\boldsymbol{w_0}}{\|\boldsymbol{w_0}\|}, \frac{b_0}{\|\boldsymbol{w_0}\|} \right) \tag{35}$$

Intuitively, *Hard-SVM* searches for $\boldsymbol{w}$ of minimal norm among all the vectors that separate the data but, if we apply the constraint exposed in (34) we enforce the margin to be 1, on the other side, now the units in which we measure the margin scale with the norm of $\boldsymbol{w}$. In other words, finding the half-space with the larges margin is the same as finding $\boldsymbol{w}$ whose norm is minimal.

5.2.2 Fritz John optimality conditions and "Support Vectors"

According to what we wrote before, we never explained why the name of the tool we are analyzing is "SVM". In other words, what are "Support vectors"? Well, using Fritz John's optimality conditions, we can state that, given $\boldsymbol{w_0}$ (34) and let $I = \{i : |\langle \boldsymbol{w_0}, \boldsymbol{x_i} \rangle| = 1\}$, there exist coefficients $\alpha_1, ..., \alpha_m$ such that:

$$\boldsymbol{w_0} = \sum_{i \in I} \boldsymbol{x_i} \alpha_i \tag{36}$$

From this fact, we can realize that $\boldsymbol{w_0}$, the solutions of the problem (34), is supported by the examples that are exactly at distance $\frac{1}{\|\boldsymbol{w_0}\|}$ from the separating hyperplane. The examples $\{\boldsymbol{x_i} : i \in I\}$ are what we call *Support Vectors*.

5.2.3 Soft SVM

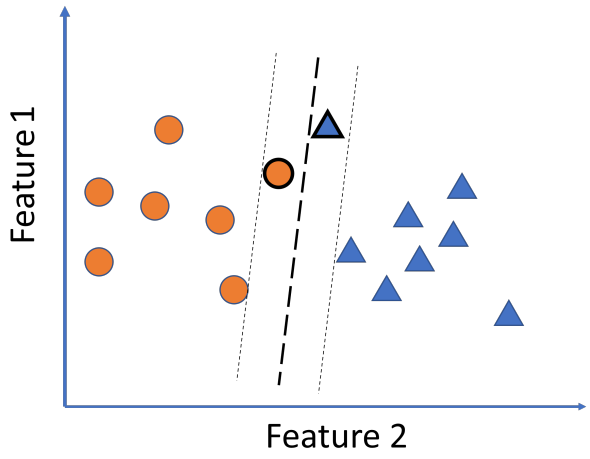

Figure 28: The main idea behind soft SVM. Note that, in this case, some points are beyond the margins.

In most of the cases, unfortunately, perfect linear separability of the data is not guarantee and, so, Hard-SVM is not applicable. What to do in such cases? Well, the most intuitive idea is to "soften" the margin and let some points lie on the borders, as shown in Fig.28. Doing so, we "relax" the rule of Hard-SVM and make this kind of tool suitable even for training sets with no

linearly separable data. Formally the problem we aim to solve becomes:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \left(\lambda \left\| \boldsymbol{w}^2 \right\| + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right) \quad s.t. \quad \forall i,\ y_i(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b) \geq 1 - \xi_i \quad and \quad \xi_i \geq 0 \tag{37}$$

We soften the margin introducing non-negative slack terms, $\xi_1, ..., \xi_m$, and what we want to do is to minimize the norm of **w** (i.e. the margin) and the average of the "violations" $\frac{1}{m} \sum_{i=1}^{m} \xi_i$. In the end, a parameter $\lambda > 0$ is introduced to balance the trade-off between these two terms. The problem (37) can be re-written if we consider the definition of *Hinge-loss* for a training-set S:

$$l_i^{hinge}((\boldsymbol{w}, b), (\boldsymbol{x_i}, y_i)) = \max \{0; 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b)\} \tag{38}$$

$$L_S^{hinge}(\boldsymbol{w}) = \frac{1}{m} \sum_{i}^{m} l_i^{hinge} \tag{39}$$

Noting that ξ_i must be non-negative, the best assignment to it would be 0 if $y_i(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b) \geq 1$ and would be $1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b)$ otherwise, we can state:

$$\xi_i = l^{hinge}((\boldsymbol{w}, b), (x_i, y_i)) \tag{40}$$

Our problem becomes, in the end:

$$\min_{\boldsymbol{w}} \lambda \left\| \boldsymbol{w}^2 \right\| + L_S^{hinge}(\boldsymbol{w}) \tag{41}$$

5.2.4 Kernel's trick

Soft SVM is able to handle noises and outliers when data are almost linear separable but, in some cases, they are not linearly separable at all. In such a condition, even softening margins could be useless. Well, the idea to handle the problem is to map our data into an higher dimension space where they are linearly separable (Fig.29).

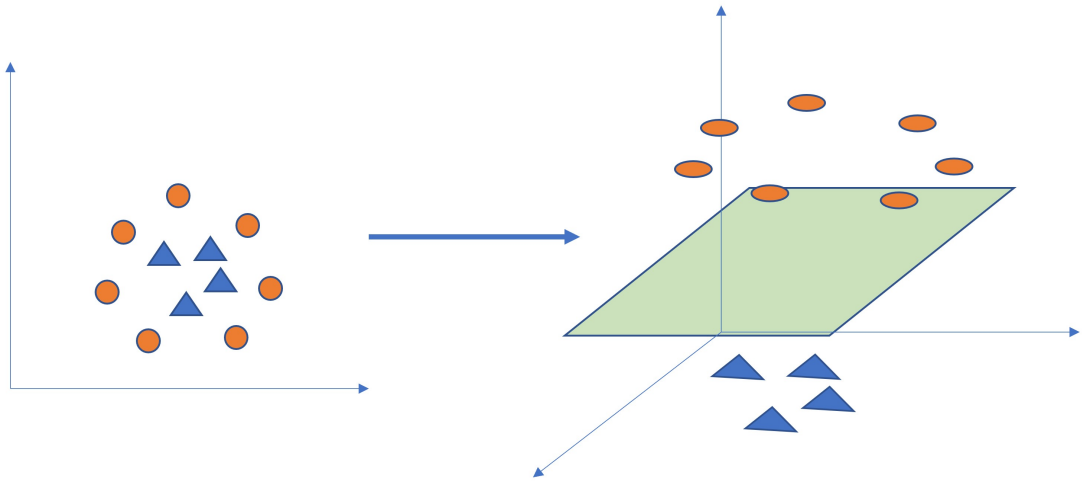

Figure 29: The main idea behind Kernels trick. The green plane is now able to separate data in a higher dimensions space.

More specifically, given the Dual Problem of (33), we want to solve:

$$\max_{\boldsymbol{\alpha}\in R^m:\boldsymbol{\alpha}\geq 0}\left(\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j\langle \boldsymbol{x_i},\boldsymbol{x_j}\rangle\right) \tag{42}$$

Since the dual problem involves only inner products, it can be seen as a $linear\ kernel$. Then, if we consider a linear mapping $\phi : X \to F$, where F is an Hilbert Space, we can define the kernel as:

$$K(\boldsymbol{x},\boldsymbol{x'})=\langle\phi(\boldsymbol{x}),\phi(\boldsymbol{x'})\rangle \tag{43}$$

Doing that, the dual problem (42) becomes:

$$\max_{\boldsymbol{\alpha}\in R^m:\boldsymbol{\alpha}\geq 0}\left(\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j\langle \phi(\boldsymbol{x_i}),\phi(\boldsymbol{x_j})\rangle\right) \tag{44}$$

The most common functions that are used as kernels are the following:

- **Polynomial kernel:** $K(\boldsymbol{x},\boldsymbol{x'})=(\langle\boldsymbol{x},\boldsymbol{x'}\rangle+1)^p$
- **RBF kernel:** $K(\boldsymbol{x},\boldsymbol{x'})=e^{-\gamma(\boldsymbol{x}-\boldsymbol{x'})^2}$

Of course, these are not all the possible kernels. In fact, according to *Mercer's Theorem*, a necessary and sufficient condition for a function to be a kernel function, is that the Kernel matrix (i.e. Gram Matrix) is positive semidefinite.

5.3 Decision Trees

Formally speaking, a decision tree is a predictor, $h : X \to Y$, that predicts the label associated with an instance $\boldsymbol{x}$ by traveling from a root node of a tree to a leaf. In our case, we focus on the binary classification setting, ($Y \in \{0; 1\}$), but decision trees can be applied for other prediction problems as well. In other words, decision trees are a classifier which is based on splitting the features' space into non overlapping distinct regions $R_1, R_2, ...R_n$ and for every observation that falls into the region R_j, we make the same prediction. Doing so, the model is able to generate the label for the unknown $\boldsymbol{x}$ point using the tree which is formed through such splits. In addition, decision trees are a really fashionable tool if we are interested in understandability and explainability of the model. In Fig.30 we can see the idea behind decision trees classifiers.

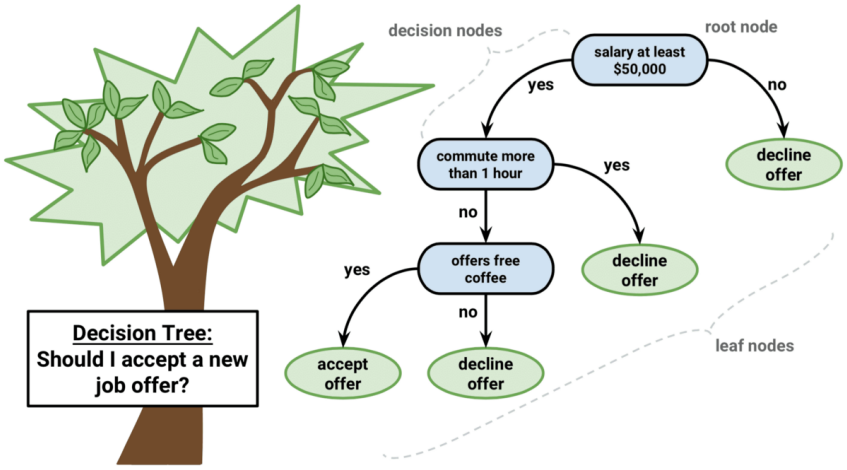

Figure 30: Idea behind decision tree. Given a binary classification task (decline or accept job offer), the trees splits over the features in order to provide a label. The feature "salary" is a numerical feature, while the feature "free coffee" is a categorical one (yes or no).

As we mentioned before, the splitting are an iterative method: at each iteration, we should split until the leaves are pure, but that is not possible in practice. As a matter of fact, acting like this, would produce a tree which is very large and may suffer from overfitting. To avoid such a problem, usually we prune the tree, setting a limit to its depth. Now let's wonder: "following which criteria do we split?" Well, we need a concept: *information gain* (IG). Given a binary decision tree, we define the IG at each split as

$$IG(D_p, f) = I(D_p) - \sum_{j \in \{left, right\}} \frac{N_j}{N_p} I(D_j) \tag{45}$$

where f is the feature we want to perform the split on, D_p and D_j are the dataset of the parent and j-th child nodes, I an *impurity measure* and N_p is the total number of training examples at the parent node, and N_j is the number of examples in the j-th child node. So, we could say IG is a difference between parent node's impurity and the weighted average of the child node impurities. We need to maximize IG at each split because we want to get the maximum gain at each split. Most of the already-implemented algorithms of decision trees, such as sklearn's one,

use two splitting criteria: *GINI* and *Entropy*.

$$GINI(t) = \sum_{i=1}^{c} p(i|t)(1 - p(i|t)) \tag{46}$$

$$Entropy(t) = -\sum_{i=1}^{c} p(i|t) log_2 p(i|t) \tag{47}$$

where $p(i|t)$ is the proportion of the examples that belong to class i for a particular node t.

Despite Decision Trees have high interpretability, they have not top performances with respect to other models. In addition, this kind of model suffers from *high variance*. For these reasons, Random Forest are used usually.

5.4 Random Forest

The idea behind such an algorithm is to combine different Decision trees (that's why *Forest*), in order to obtain better performances. We take random samples with replacement from our dataset to generate different bootstrapped training sets. In this way we can build several decision trees whose predictions we can aggregate with a majority voting procedure. This approach is called *bootstrap aggregation*. In addition to that, in Random Forest Algorithm, what we try to do is decorrelate the trees and, to do that, we use *feature bagging*. At each time a split in a tree is considered, we select a fixed number of features, randomly. Usually the number of randomly selected features is $\sqrt{p_{tot}}$ where p_{tot} is the total number of features. The final prediction will be given by a majority voting scheme of all the estimators. Figure 31 displays a graphical idea of such an approach.

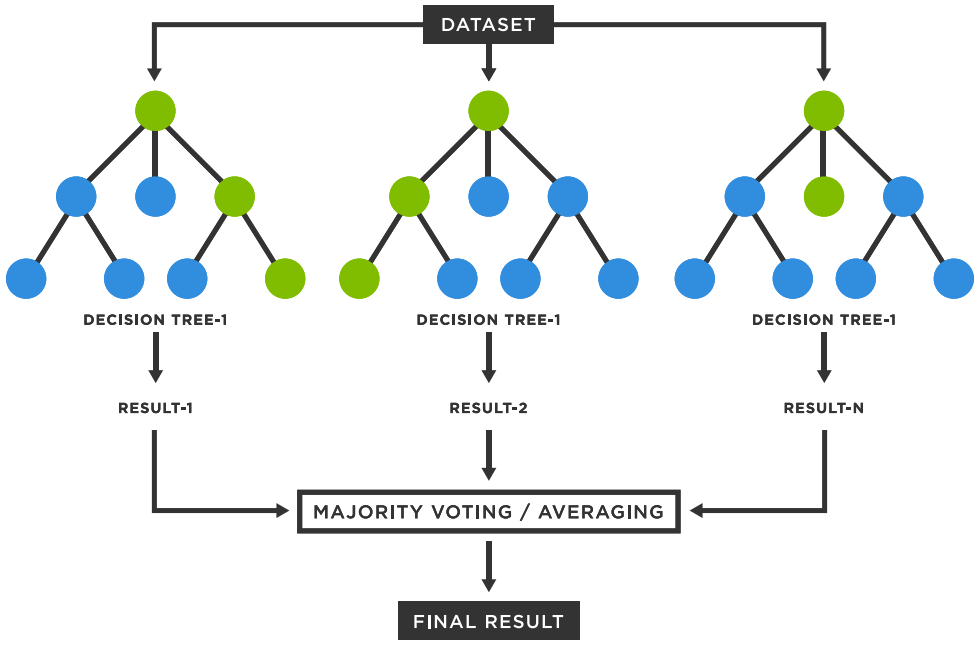

Figure 31: Idea behind Random Forest

5.5 KNN

Nearest Neighbor algorithms are among the simplest of all machine learning algorithms. The idea is to memorize the training set and then to predict the label of any new instance on the basis of the labels of its closest neighbors in the training set. Let's now try to explain how it works.

Given a positive integer k and a test observation x_0, KNN identifies the the k closest points to x_0 (N_0) and then, it estimates the conditional probability for class j as the fraction of points in N_0 whose target value is equal to j:

$$P(Y = j|X = x_0) = \frac{1}{k} \sum_{i \in N_0} \mathbb{1}_{y_i = j} \tag{48}$$

Then x_0 is classified with the class j having the highest conditional probability. As it's possible to imagine, the most important hyper-parameters to tune are k and the adopted distance computation algorithm.

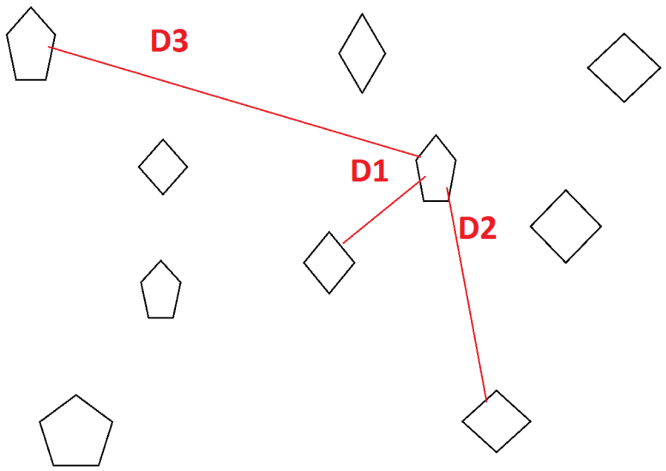

Figure 32: KNN main idea. Distance between points is computed to understand which are the nearest neighbors

5.6 Models Performance comparison

In the following pages, we give an overview on the results we managed to obtain for each classifier. Different preprocessing combinations were tested: with or without outlier removal, applying dimensionality reduction techniques (LDA or PCA) or not, using or not different resampling techniques. The metric we choose to adopt is F1-score. Precision-recall curve and Confusion Matrix of the best model have been provided for each model.

5.6.1 Logistic Regression

According to Fig.33, it's easy to notice how the best performances have been obtained using LDA and SMOTE. Moreover, the application of LDA lead us to obtain better results with respect to application of PCA. Moreover, confusion matrix and precision-recall curve for best model have been provided in Fig.34.

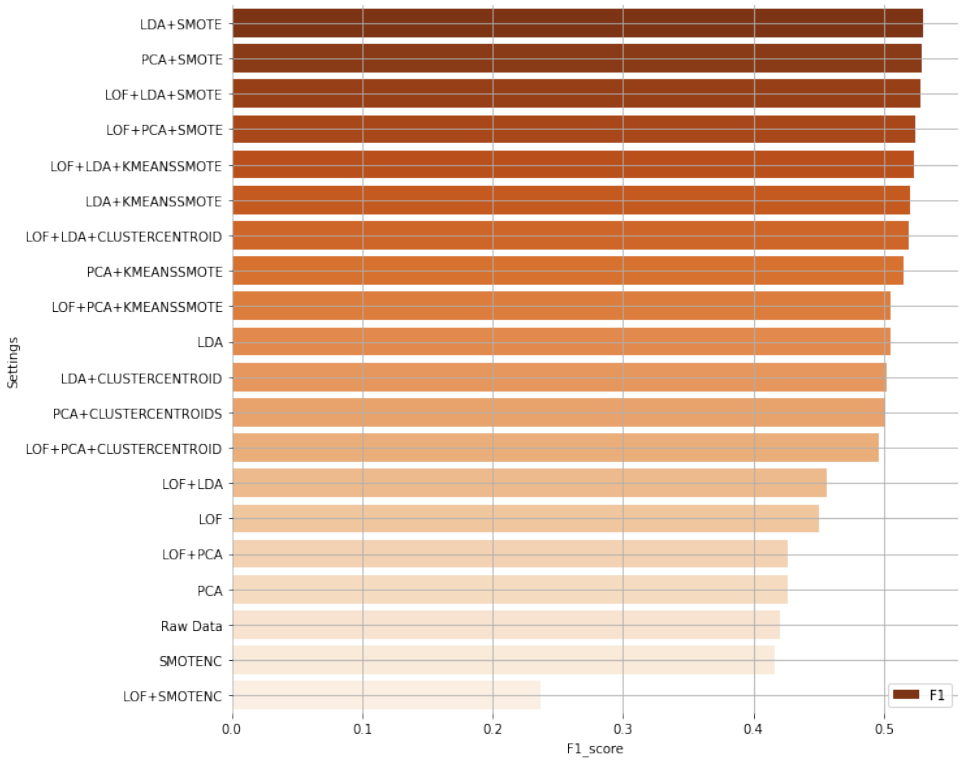

Figure 33: Results obtained for Logistic Regression trained in different settings.

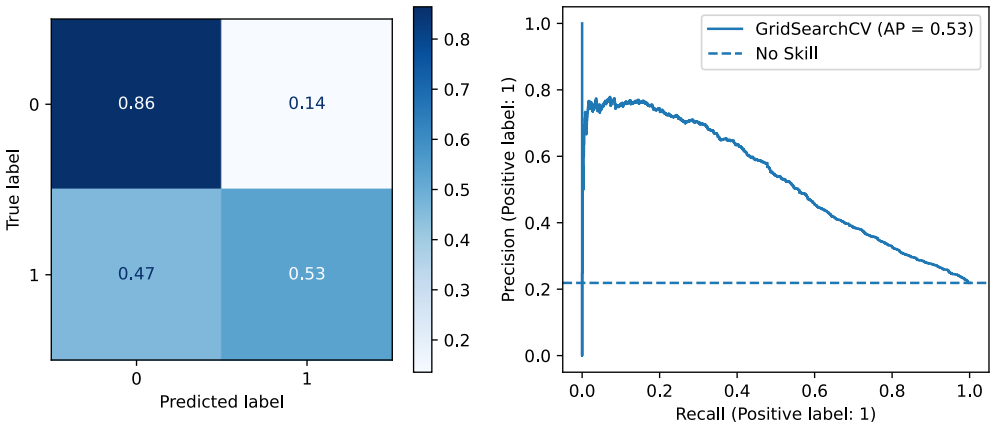

Figure 34: Confusion matrix and precision-recall curve for the best settings for Logistic Regression mdoel.

5.6.2 SVM

According to Fig.35, it's easy to notice how the best performances have been obtained using LDA and SMOTE in combination with the application of LOF outlier detection technique. Moreover, the application of LDA lead us to obtain better results with respect to application of PCA. As we made before, confusion matrix and precision-recall curve for best model have been provided in Fig.36.

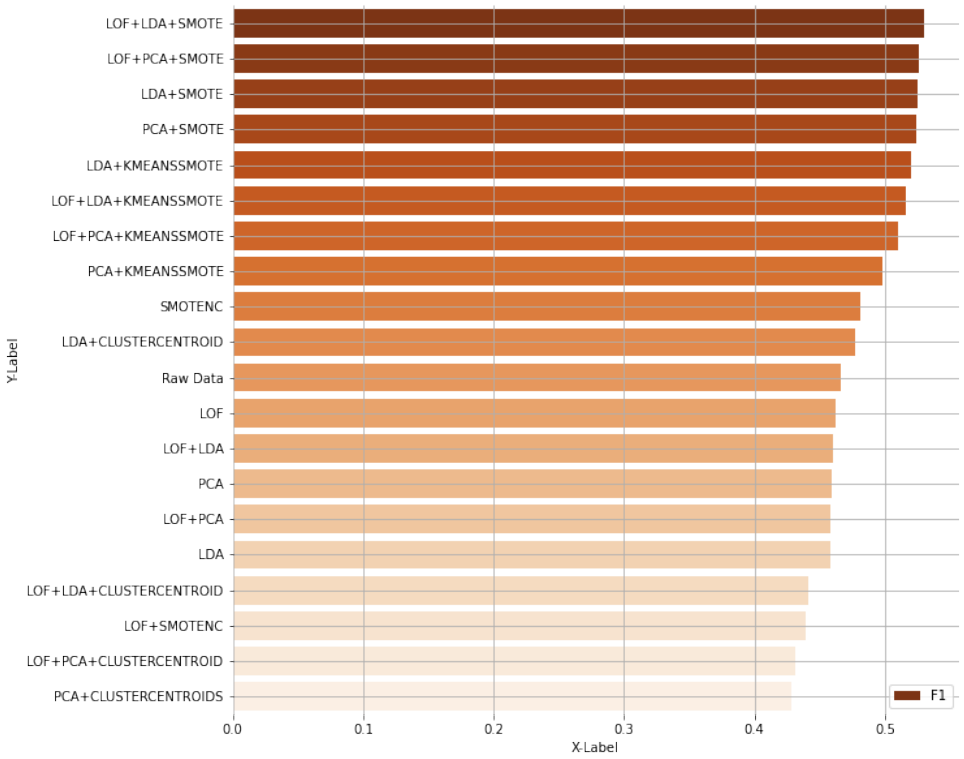

Figure 35: Results obtained for SVM Classifier using different settings

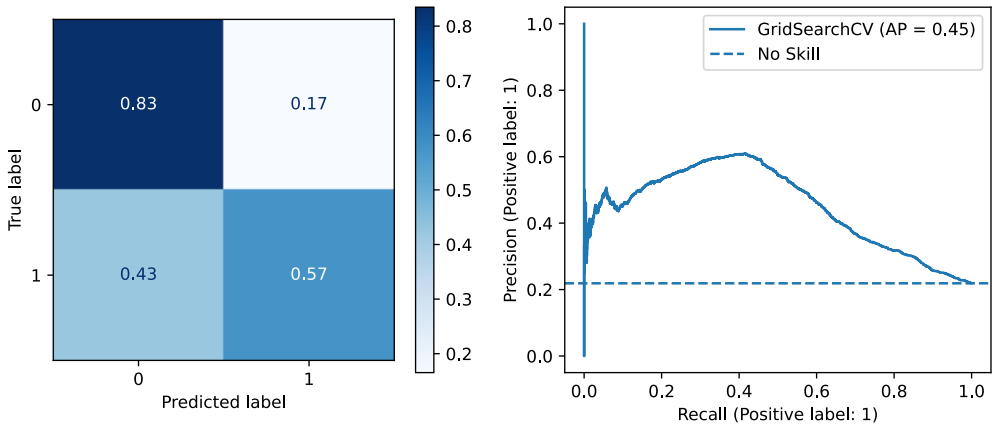

Figure 36: Confusion matrix and precision-recall curve for the best settings for SVM.

5.6.3 Decision Tree

According to Fig.37, it's easy to notice how the best performances have been obtained using LDA and SMOTE in combination with LOF outlier removal. Moreover, the application of LDA lead us to obtain better results with respect to application of PCA. Confusion matrix and precision-recall curve for best model have been provided in Fig.38.

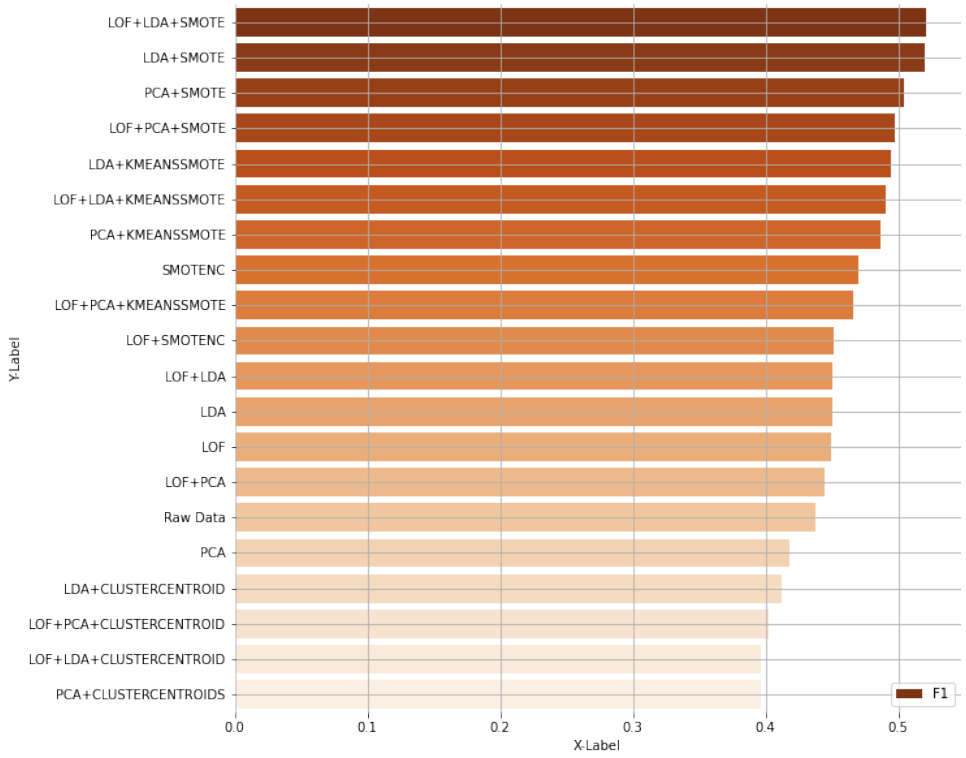

Figure 37: Results obtained for Decision Tree Classifier trained in different settings

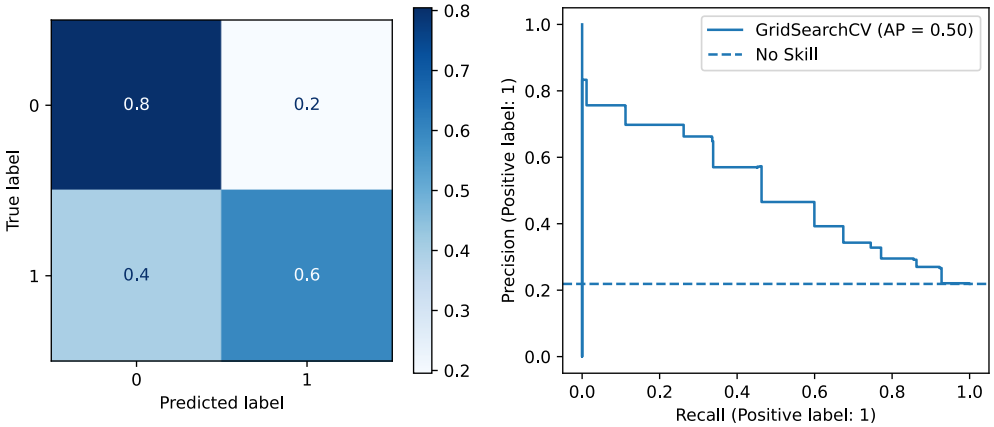

Figure 38: Confusion matrix and precision-recall curve for the best settings for Decision Tree model.

5.6.4 Random Forest

According to Fig.39, it's easy to notice how the best performances have been obtained using PCA and k-means SMOTE. Moreover, the application of PCA lead us to obtain better results with respect to application of LDA. In Fig.40, confusion matrix and precision-recall curve for best model are displayed.

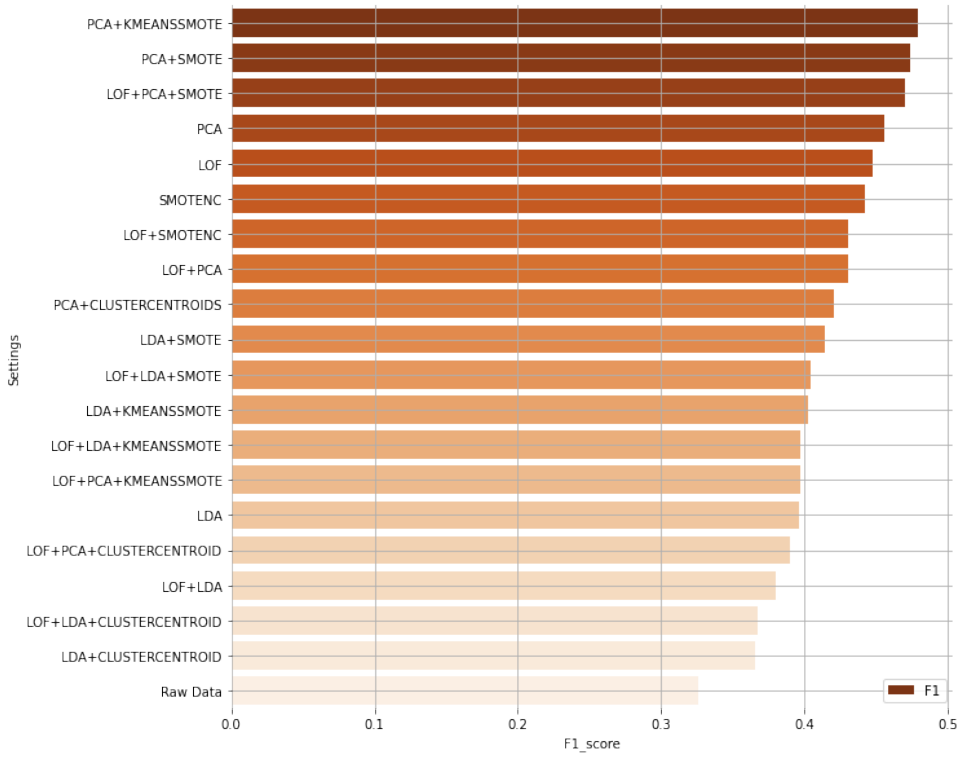

Figure 39: Results obtained for Random Forest trained in different settings

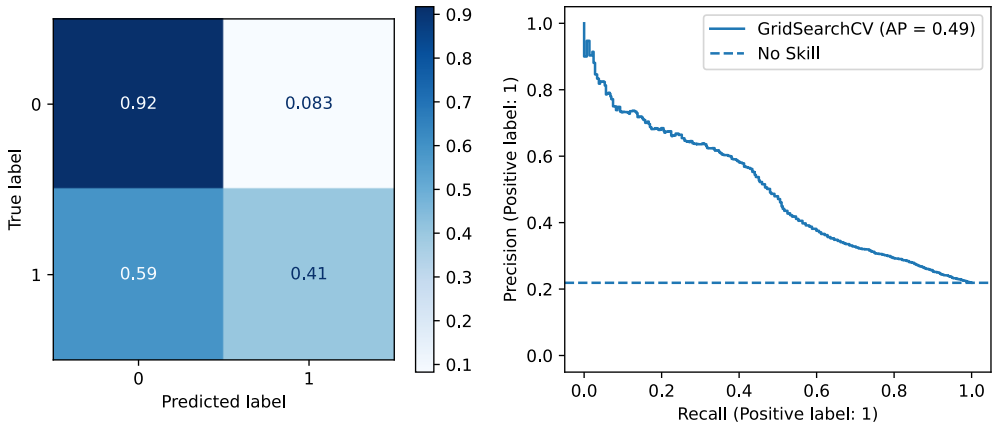

Figure 40: Confusion matrix and precision-recall curve for the best settings for Random Forest model.

5.6.5 K Nearest Neighbors

In case of KNN, best results were obtained combining LDA algorithm and SMOTE resampling techinque. In Fig.41, a comparison between different results obtained with different techniques is provided. Furthermore, as we made before, confusion matrix and precision-recall curve for best model have been provided in Fig.42.

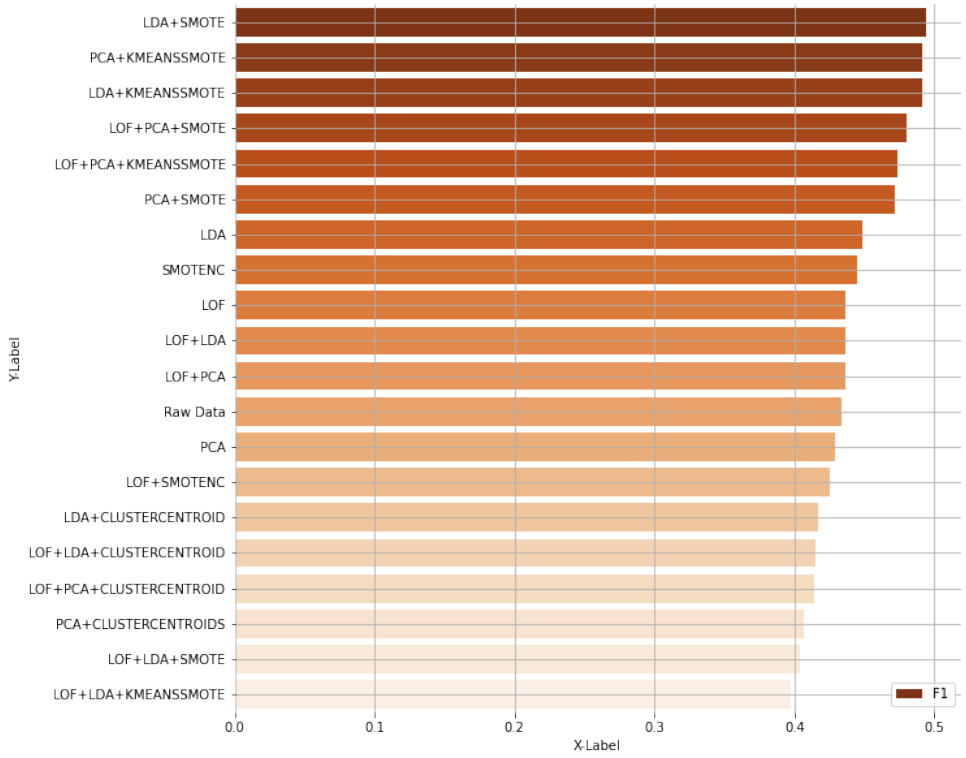

Figure 41: Results obtained for KNN Classifier in different settings

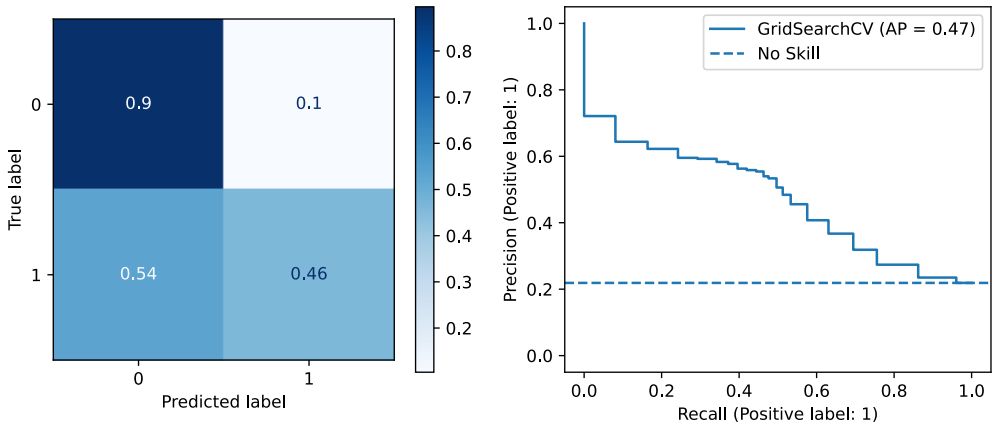

Figure 42: Confusion matrix and precision-recall curve for the best settings for KNN Classifier.

6 Conclusions

During this project, we have tested different Machine Learning models on an unbalanced dataset dealing with a binary classification task. This *Tesina* helped us to understand how to perform an entire machine learning analysis using a custom pipeline. Moreover, it taught us how to deal with unbalanced datasets and how to apply all the steps of a pipeline in combination with Cross Validation technique. In the end, we managed to obtain quite good F1 scores, despite the fact the dataset was unbalanced.

References

[1] I. Yeh and C.-H. Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert Systems with Applications*, vol. 36, pp. 2473–2480, 03 2009.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, p. 93–104, may 2000. [Online]. Available: https://doi.org/10.1145/335191.335388

[3] T. Yiu, "The curse of dimensionality," Sep 2021. [Online]. Available: https://towardsdatascience.com/the-curse-of-dimensionality-50dc6e49aa1e

[4] G. Lemaitre, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," 2016. [Online]. Available: https://arxiv.org/abs/1609.06570

[5] G. Douzas, F. Bacao, and F. Last, "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE," *Information Sciences*, vol. 465, pp. 1–20, oct 2018. [Online]. Available: https://doi.org/10.1016%2Fj.ins.2018.06.056

[6] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning - From Theory to Algorithms.* Cambridge University Press, 2014.

[7] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 1137–1143.

[8] H. He and E. Garcia, "Learning from imbalanced data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 9, pp. 1263–1284, Sept 2009. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5128907&tag=1

[9] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R.* Springer, 2013. [Online]. Available: https://faculty.marshall.usc.edu/gareth-james/ISL/